У меня написано и опубликовано много статей по управлению современным веб сервером. Сегодня я подробно расскажу, как установить и выполнить настройку nginx с подробным описанием функционала и примерами. Я постарался охватить все наиболее актуальные и полезные возможности бесплатной версии nginx.

Содержание:

Введение

Базовая настройка nginx не представляет каких-то сложностей на первом этапе. У nginx есть неплохая документация на русском языке. Еще лучше на английском. Практически все, что я буду рассказывать, есть там. Понятное дело, что если вы только знакомитесь с продуктом, или используете его базовый функционал, копать документацию на начальном этапе будет не очень продуктивно. Моя статья будет служить беглым обзором основных возможностей, которые я сам использовал, с моими же примерами.

По тематике некоторых разделов у меня уже есть подробные статьи. В таком случае я буду давать на них ссылки с краткими комментариями здесь. В конце статьи приведу свой типовой конфиг для nginx, который я обычно беру за основу, когда настраиваю очередной web сервер.

Установка nginx

В своей статье я не буду привязываться к конкретному дистрибутиву, так как настройка nginx одинакова везде. Формат файла один и тот же, поэтому конфигурация без проблем может мигрировать на разные системы. Править придется только пути к файлам и директориям. Тем не менее, я расскажу, как выполнить установку на популярные дистрибутивы.

Установку nginx на CentOS 7 я подробно разбирал в соответствующей статье про настройку web сервера на centos. Здесь просто перечислю необходимые шаги.

Подключаем репозиторий nginx для CentOS 7:

```
# rpm -Uvh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7.ngx.noarch.rpm
```

Выполняем установку:

```
# yum install nginx
```

Вот и все. На этом установка на Centos 7 закончена. Пример установки nginx на Debian.

Подключаем репозиторий для Debian:

```
# echo "deb http://nginx.org/packages/debian `lsb_release -cs` nginx" | tee /etc/apt/sources.list.d/nginx.list
```

Импортируем ключ для проверки подлинности пакетов:

```
# curl -fsSL https://nginx.org/keys/nginx_signing.key | sudo apt-key add -
```

Устанавливаем nginx на Debian:

```
# apt update && apt install nginx
```

Установим nginx на Ubuntu. Подключаем репозиторий:

```
# echo "deb http://nginx.org/packages/ubuntu `lsb_release -cs` nginx" | tee /etc/apt/sources.list.d/nginx.list
```

Импортируем ключ:

```
# curl -fsSL https://nginx.org/keys/nginx_signing.key | sudo apt-key add -
```

Выполняем установку nginx на Ubuntu:

```
# apt update && apt install nginx
```

На всех указанных системах запуск веб сервера выполняется командой:

```
# systemctl start nginx
```

Добавляем nginx в автозагрузку:

```
# systemctl enable nginx
```

В последнее время большое распространение получили контейнеры, в частности docker. Довольно популярна ситуация, когда nginx работает в докере, поэтому отдельно рассмотрю вопрос установки nginx в docker, хоть там и нет ничего сложного.

Nginx в Docker

Неоспоримые преимущества от использования docker получают разработчики, поэтому они очень часто его используют. В том числе в виде контейнеров docker с nginx. Установка nginx через docker может быть выполнена из официального образа nginx в Docker Hub. Установить и запустить nginx в docker можно примерно такой командой:

```
# docker run --name nginx01 -p 80:80 -d nginx
```

В данной команде:

- *nginx01* — имя созданного контейнера, основанного на базовом образе nginx
- *-p 80:80* — сопоставление порта на локальной машине, порту в контейнере. Сначала указывается локальный порт, потом внутри контейнера.
- *-d* — ключ, обозначающий запуск контейнера в режиме службы.

Это общий случай установки. Образ nginx в данном случае будет скачан автоматически при создании первого контейнера. Для удобства используется большее количество параметров. Для примера запустим еще один контейнер с другим названием и с расширенными параметрами.

```
# docker run --name nginx02 -p 81:80 -v ~/nginx/www:/usr/share/nginx/html -v ~/nginx/conf:/etc/nginx -v
~/nginx/logs:/var/log/nginx -d nginx
```

В данном примере мы создали еще один контейнер naginx02, назначили ему порт 81 на локальной машине, в контейнер смонтировали 3 локальные директории:

1. *~/nginx/www* — здесь будут лежать исходники сайта.
2. *~/nginx/conf* — полная конфигурация nginx. Ее нужно будет скопировать откуда-то.

3. *~/nginx/logs* — логи nginx.

Не обязательно выносить на хост все эти папки. Я показываю для примера. Вы выбирайте только то, что вам действительно нужно.

Таким образом можно легко работать с nginx с помощью docker. Можно без проблем запустить сколько угодно контейнеров с nginx, указав каждому свой порт, конфигурацию и директорию с исходниками. Для разработки это действительно удобно. Для продакшена отдельный разговор. Те, кто используют nginx в docker в production вряд ли нуждаются в данной статье.

Рестарт nginx и другие параметры командной строки

Перед тем, как двигаться дальше к настройке nginx, предлагаю пройтись по основным параметрам в командной строке. Это упростит и ускорит дальнейшую работу.

Прежде всего расскажу, как перезапустить nginx. С помощью systemctl на всех дистрибутивах это выглядит одинаково.

```
# systemctl restart nginx
```

Перед перезагрузкой nginx, рекомендую выполнить проверку конфигурации:

```
# nginx -t
```

Еще одна важная команда, с помощью которой можно применить новую конфигурацию nginx без остановки и перезапуска веб сервера. Будет запущен новый рабочий процесс с новой конфигурацией, а старые процессы плавно завершатся.

```
# nginx -s reload
```

Следующая команда помимо тестирования конфигурации, выводит полный конфиг на экран. Вывод можно направить в отдельный файл и там проанализировать. Это удобно, когда у вас конфигурация состоит из множества вложенных конфигов, правильность которых трудно оценить разом.

```
# nginx -T
```

```
root@ubuntu16:~# nginx -T
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
# configuration file /etc/nginx/nginx.conf:

user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;
```


Так же бывает полезно посмотреть полную информацию о версии nginx, параметрах сборки, модулях и т.д.

```
# nginx -V
```

```
root@ubuntu16:/etc/nginx# nginx -V
nginx version: nginx/1.14.2
built by gcc 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.9)
built with OpenSSL 1.0.2g  1 Mar 2016
TLS SNI support enabled
configure arguments: --prefix=/etc/nginx --sbin-path=/usr/sbin/nginx --modules-path=/usr/lib/nginx/modules --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 --lock-path=/var/run/nginx.lock --http-client-body-temp-path=/var/cache/nginx/client_temp --http-proxy-temp-path=/var/cache/nginx/proxy_temp --http-fastcgi-temp-path=/var/cache/nginx/fastcgi_temp --http-uwsgi-temp-path=/var/cache/nginx/uwsgi_temp --http-scgi-temp-path=/var/cache/nginx/scgi_temp --user=nginx --group=nginx --with-compat --with-file-aio --with-threads --with-http_addition_module --with-http_auth_request_module --with-http_dav_module --with-http_flv_module --with-http_gunzip_module --with-http_gzip_static_module --with-http_mp4_module --with-http_random_index_module --with-http_realip_module --with-http_secure_link_module --with-http_slice_module --with-http_ssl_module --with-http_stub_status_module --with-http_sub_module --with-http_v2_module --with-mail --with-mail_ssl_module --with-stream --with-stream_realip_module --with-stream_ssl_module --with-stream_ssl_preread_module --with-cc-opt='-g -O2 -fstack-protector-strong -Wformat -Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -fPIC' --with-ld-opt='-Wl,-Bsymbolic-functions -Wl,-z,relro -Wl,-z,now -Wl,--as-needed -pie'
root@ubuntu16:/etc/nginx#
```


Например, мне эта информация была нужна, когда я делал собственную сборку nginx с поддержкой tls 1.3 и модулем сжатия brotli.

В принципе, на этом все. Не припоминаю, чтобы я использовал что-то еще. Плавно переходим к конфигурации nginx.

Виртуальные хосты

После дефолтной установки nginx у вас уже будет один виртуальный хост, который описан конфигом *default.conf*. Обычно конфиги с виртуальными хостами расположены в директории */etc/nginx/conf.d*. Здесь, в отличие от Apache, структура размещения конфигурационных файлов одинаковая на всех дистрибутивах, что очень удобно.

Что такое виртуальный хост? Попробую объяснить своими словами. Это конфигурационный файл, который описывает настройку условно одного домена. Для удобства, каждый виртуальных хост выносят в отдельный конфиг, но никто не мешает вам все это описывать в общем конфигурационном файле. Дело в том, что если доменов у вас много, то работать в одном большом конфиге не удобно, поэтому я рекомендую вам придерживаться схемы файлов конфигурации, когда один конфиг описывает параметры одного домена.

Виртуальные хосты наследуют параметры из основного файла конфигурации nginx.conf, но эти параметры могут быть переопределены в конкретном виртуальном хосте. То есть можно задать дефолтные параметры для всех сайтов, но в случае необходимости переопределить какой-то параметр в конкретном виртуальном хосте. Вот типичный пример конфига.

```
server {
        listen 80;
        server_name example.com www.example.com;
        access_log /var/log/nginx/example.com.access.log main;
        root   /var/www/example.com/public;
```

```
        location / {
                index  index.html index.htm index.php;
        }
        location ~ \.php$ {
                fastcgi_pass   127.0.0.1:9000;
                fastcgi_index  index.php;
                fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
                include        fastcgi_params;
        }
}
```

Все не указанные явно в этом виртуальном хосте параметры будут унаследованы из основного файла конфигурации *nginx.conf*. Допустим, вам надо по какой-то причине отключить сжатие gzip на этом хосте и указать кодировку windows-1251. Тогда конфиг приобретет следующий вид.

```
server {
        listen 80;
        server_name example.com www.example.com;
        access_log /var/log/nginx/example.com.access.log main;
        root   /var/www/example.com/public;
        charset windows-1251;
        gzip off;

        location / {
                index  index.html index.htm index.php;
        }
        location ~ \.php$ {
                fastcgi_pass   127.0.0.1:9000;
                fastcgi_index  index.php;
                fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
                include        fastcgi_params;
        }
}
```

С помощью виртуальных хостов в nginx вы можете очень гибко настраивать конфигурацию каждого домена. Как минимум я рекомендую для каждого домена делать отдельно:

- директорию с исходниками сайта;
- директорию с логами;
- в некоторых ситуациях отдельный php-fpm пул для каждого сайта или группы сайтов.

Более полный пример настройки виртуального хоста для реального сайта на wordpress смотрите в отдельной статье по настройке web сервера. Мы же переходим к настройке location.

Настройка location в конфигурации

Как вы могли заметить, в предыдущем примере я использовал 2 разных location для виртуального хоста. Постараюсь простыми словами рассказать, что это такое, как использовать и для чего вообще нужно.

С помощью location в nginx вы можете управлять конфигурацией в зависимости от URI запроса. Если в виртуальных хостах мы могли переопределять настройки в зависимости от имени домена, то тут мы спускаемся на уровень ниже и управляем параметрами в зависимости от пути запроса. В примере с виртуальными хостами у меня было 2 location:

1. / — корень сайта, перехватывает вообще все запросы к домену.
2. ~ \.php$ — запросы к файлам с расширением php. То есть если в запросе указан путь к файлу .php, то к нему применяется параметр fastcgi_pass и запрос отправляется на обработку к php-fpm.

Location можно задавать префиксной строкой или регулярным выражением. В регулярных выражениях используются модификаторы ~, либо ~* Без звездочки учитывается регистр, со звездочкой нет. Обработка location в конфигурационном файле идет в следующей последовательности:

1. Первыми проверяются префиксные строки. Совпадения запоминаются.
2. Дальше проверяются регулярные выражения в том порядке, как они перечислены в конфигурационном файле. Проверка по регулярным выражениям прекращается сразу же после совпадения. Если совпадение не было найдено, используется запомненный ранее префикс.

Так же в location может использоваться префикс = Он означает точное совпадение запроса и заданного location. После такого совпадения, остальные проверки сразу же прекращаются. Рекомендуется использовать этот префикс, если у вас огромное количество конкретных запросов. Используя префикс = для них, вы снизите нагрузку на сервер, так как запросы не будут проверяться по всем правилам.

В крупном сайте может быть огромное количество различных location. Вот несколько простых реальных примеров из моей практики.

На своем сайте я решил отказаться от использования amp страниц, выключил их и сделал перенаправление с этих страниц на обычные. В данном примере amp располагались по исходному урлу с добавлением /amp/ в конец пути.

```
location ~ /amp/$ {
rewrite ^(.*/)amp/$ $1 permanent;
}
```

В этом примере укажем максимальный срок хранения картинок и шрифтов в кэше а так же отключим для них логирование.

```
location ~* ^.+.(js|css|png|jpg|jpeg|gif|ico|woff|woff2|swf|ttf|svg)$ {
access_log off;
expires 1y;
}
```

Закроем доступ к директории *.git* на сайте.

```
location ~ /.git {
return 404;
}
```

Запретим исполнение скриптов в перечисленных директориях.

```
location ~* /(images|cache|media|logs|tmp)/.*.(php|pl|py|jsp|asp|sh|cgi)$ {
return 404;
}
```

И так далее. Думаю, смысл понятен. Примеров может быть огромное количество. Location важный параметр конфигурации в настройке nginx. Немного информации на эту тему можно посмотреть в соответствующем разделе документации.

Работа nginx с php-fpm

В предыдущих разделах я уже показал примеры конфигурации, где запросы по определенным URI перенаправляются на php-fpm. Еще более подробно я рассмотрел этот вопрос в отдельной статье по настройке php-fpm. Сейчас просто покажу на реальном примере, как выглядит взаимодействие nginx и php-fpm.

Php-fpm может слушать как сокет unix, так и tcp порт. Эти настройки задаются в конфиге пула. Это может быть либо

```
listen = 127.0.0.1:9000
```

либо

```
listen = /var/run/php-fpm/php-fpm.sock
```

В зависимости от того, в каком режиме работает php-fpm, зависят настройки в nginx.

Вот примерный конфиг php-fpm для пула www.conf на виртуальной машине с 1Gb памяти.

```
[www]
listen = /var/run/php-fpm/php-fpm.sock
listen.allowed_clients = 127.0.0.1
listen.mode = 0660
listen.owner = nginx
listen.group = nginx
user = nginx
group = nginx
; как будут создаваться новые рабочие процессы
pm = dynamic
; максимальное количество рабочих процессов
pm.max_children = 15
; число запущенных процессов при старте сервера
pm.start_servers = 6
```

```
; минимальное и максимальное количество процессов в простое
pm.min_spare_servers = 4
pm.max_spare_servers = 8
slowlog = /var/log/php-fpm/www-slow.log
pm.max_requests = 250
php_admin_value[error_log] = /var/log/php-fpm/www-error.log
php_admin_flag[log_errors] = on
php_value[session.save_handler] = files
php_value[session.save_path] = /var/lib/php/session
pm.status_path = /status
```

Чтобы заработал php в nginx через php-fpm, достаточно убедиться, что php-fpm работает и указать в виртуальном хосте location для php. Пример реальных настроек для wordpress сайта.

```
location ~ \.php$ {
 try_files  $uri =404;
 fastcgi_pass   unix:/var/run/php-fpm/php-fpm.sock;
 #fastcgi_pass    127.0.0.1:9000;
 fastcgi_index index.php;
 include fastcgi_params;
 fastcgi_param DOCUMENT_ROOT /web/sites/example.com/www/;
 fastcgi_param SCRIPT_FILENAME /web/sites/example.com/www$fastcgi_script_name;
 fastcgi_param PATH_TRANSLATED /web/sites/example.com/www$fastcgi_script_name;
 fastcgi_param QUERY_STRING $query_string;
 fastcgi_param REQUEST_METHOD $request_method;
 fastcgi_param CONTENT_TYPE $content_type;
 fastcgi_param CONTENT_LENGTH $content_length;
 fastcgi_param HTTPS on; # включать, если сайт по https работает
 fastcgi_intercept_errors on;
 fastcgi_ignore_client_abort off;
 fastcgi_connect_timeout 60;
 fastcgi_send_timeout 180;
```

```
 fastcgi_read_timeout 180;
 fastcgi_buffer_size 128k;
 fastcgi_buffers 4 256k;
 fastcgi_busy_buffers_size 256k;
 fastcgi_temp_file_write_size 256k;
}
```

В целом все. Этого достаточно для настройки связки nginx + php-fpm. Типовой ошибкой в данном случае является то, что nginx не имеет доступа к unix сокету php-fpm. По-умолчанию после установки он запускается с правами apache. Если пользователя не исправить на nginx, то у веб сервера не будет доступа к сокету, php не заработает. В своем примере конфига php-fpm я указал пользователя правильно.

Настройка SSL сертификата

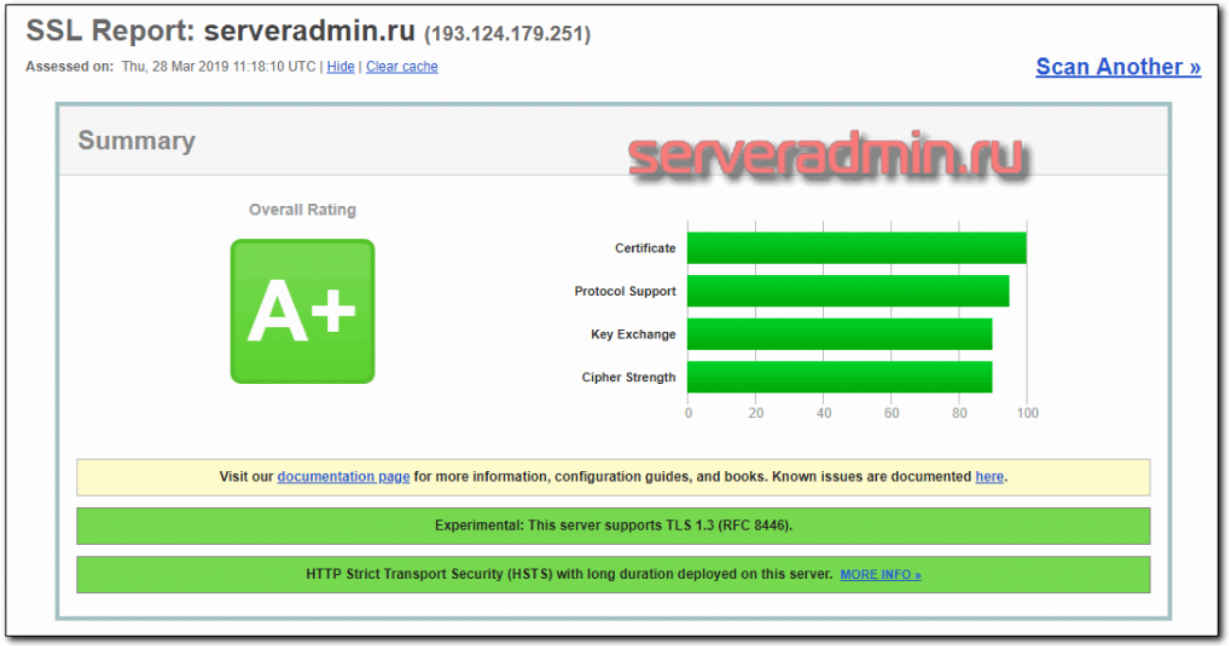
Далее рассмотрим момент с настройкой ssl сертификатов в nginx. В общем случае с этим не должно быть каких-то проблем. Тут все просто. Можно глобально задать настройки ssl для всех виртуальных хостов, а можно отдельно в каждом. Вот пример настроек ssl для *nginx.conf*.

```
ssl_session_cache shared:SSL:50m;
ssl_session_timeout 1d;
ssl_session_tickets on;
ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;
ssl_ciphers 'TLS13-CHACHA20-POLY1305-SHA256:TLS13-AES-128-GCM-SHA256:TLS13-AES-256-GCM-SHA384:ECDHE:ECDHE-ECDSA-CHACHA20-
POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-
AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-
AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-
ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-
SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS';
ssl_prefer_server_ciphers on;
ssl_dhparam /etc/ssl/certs/dhparam.pem;
ssl_stapling on;
ssl_stapling_verify on;
```

```
add_header Strict-Transport-Security max-age=15768000;
resolver 8.8.8.8;
```

Здесь уже включена поддержка TLS 1.3 и соответствующие шифры. Сейчас не готов прокомментировать именно такой выбор шифров. В свое время, перед настройкой TLS 1.3 я изучил этот вопрос и собрал такой набор, который везде использую.

Часто возникают споры насчет директивы resolver. На dns сервер 8.8.8.8 есть нарекания по стабильности. Так что выбор dns сервера остается за вами. С подобными настройками ssl вы получите рейтинг A+.



Для генерации файла **dhparam.pem**, воспользуйтесь командой:

```
# openssl dhparam -out /etc/ssl/certs/dhparam.pem 4096
```

Процесс длится долго, до получаса на слабых виртуалках. Этот файл нужен для повышения безопасности и получения максимального рейтинга. Насколько этот параметр критичен в реальности, не берусь судить. Если тороплюсь, то настраиваю без него. Подробный разбор параметров ssl можно посмотреть в статье на хабре.

В виртуальном хосте надо будет добавить настройки, касающиеся непосредственно сертификатов, а так же указать, что сервер слушает 443 порт.

```
server {
 listen 443 ssl http2;
```

```
 ......................
 ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
 ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
 ......................
  location ~ \.php$ {
   ......................
   fastcgi_param HTTPS on;
   ......................
  }
}
```

Редирект с http на https

Добавим переадресацию с http на https. Для этого добавляем в настройки виртуального хоста еще одну директиву server.

```
server {
 listen 80;
 server_name example.com www.example.com;
 access_log /var/log/nginx/example.com.access.log main;
 root   /var/www/example.com/public;

 location / {
  return 301 https://example.com$request_uri;
 }
}
```

Проксирование запросов

Nginx очень производительный веб сервер, поэтому его часто используют в качестве Reverse Proxy для других служб и серверов. Подробно вопрос проксирования запросов в nginx с помощью proxy_pass я рассмотрел отдельно. Сейчас же в двух словах объясню, что это такое. Допустим, у вас есть какой-то сервис на отдельном сервере и вы ходите перенаправлять на него часть запросов с вашего сайта. Для этого вы делаете отдельный location и

указываете, что все запросы по определенному правилу нужно перенаправлять на этот сервер.

```
location /forum/ {
 proxy_pass http://192.168.13.31;
 proxy_set_header Host $host;
 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
 proxy_set_header X-Real-IP $remote_addr;
 proxy_redirect default;
}
```

Все запросы с урлами, содержащими /forum/ будут перенаправлены на отдельный сервер, где трудится тяжелый форум со своей базой данных и своими настройками. Им может управлять другой администратор и вообще он не имеет к вам никакого отношения. Таким образом, большой проект можно разделить на части с делегированием полномочий. Но это отдельная история.

Nginx proxy_pass удобно использовать разработчикам для проксирования запросов в разные docker контейнеры, которые подняты на рабочей машине на разных портах. Можно перенаправлять не только отдельные урлы, но и весь сайт. Допустим, вы делаете какие-то фильтрации трафика на стороне сервера с nginx, а потом все запросы отправляете на исходный сайт. Тогда делаете простую настройку для проксирования:

```
location / {
proxy_pass http://192.168.13.31;
proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Real-IP $remote_addr;
 }
```

Все запросы уходят на сторонний сервер. Частным случаем проксирования в nginx является работа в связке с Apache, о чем я расскажу далее.

Nginx в связке с Apache

Популярным кейсом работы nginx является работа в качестве Reverse Proxy для Apache. Лет 5-10 назад, когда nginx был не очень распространен, это было очень популярное решение. Сейчас Nginx во многих областях полностью заменил Apache, но тем не менее, пока еще не до конца. К примеру, очень

популярный в России движок для сайтов Bitrix до сих пор требует в качестве web сервера Apache.

Покажу на простом примере, как Nginx настроить в качестве front-end к Apache. Статические данные будет обрабатывать сам nginx, а динамические запросы перенаправлять на apache.

Для начала вам необходимо настроить apache стандартным образом, с той лишь разницей, что слушать он должен не привычный 80-й или 443 порт, а, к примеру, 8080. Далее в настройках виртуального хоста указываете для каких запросов будет перенаправление на apache. Для примера отправим туда все, что не является статичным контентом. Создаем 2 location:

1. Статика, которую будет напрямую отдавать Nginx.
2. Все остальные запросы, которые будет обрабатывать Apache.

```
server {
        listen 80;
        server_name example.com www.example.com;
        access_log /var/log/nginx/example.com.access.log main;
        location ~* ^.+.(js|css|png|jpg|jpeg|gif|ico|woff|woff2|swf|ttf|svg|html|txt)$ {
                root   /var/www/example.com/public;
                expires 1y;
        }

        location / {
                proxy_pass http://127.0.0.1:8080/;
                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_connect_timeout    120;
                proxy_send_timeout    120;
                proxy_read_timeout    180;
        }
}
```

В данном примере Nginx и Apache работают на одном сервере. Но это не обязательно. Web сервер с apache вы без проблем можете разместить на другой

машине. Отдельно будет стоять вопрос определения реальных ip адресов клиентов на сервере с Apache. Я его рассмотрел подробно в статье про nginx revers proxy, ссылку на которую привел в предыдущем разделе.

Балансировка нагрузки

Nginx может выступать в качестве балансировщика. Настройку балансировки в nginx я так же подробно рассматривал отдельно. Все подробности в статье. Здесь кратко упомяну, что это такое. Nginx умеет распределять нагрузку между несколькими серверами. Правила распределения настраиваются, как и количество серверов. Покажу как это делать на предыдущем примере с apache.

Допустим, у вас очень нагруженный сайт. А нагрузить Bitrix без настроенного кэширования, очень просто. Вы хотите добавить еще один сервер с Apache, чтобы распределить нагрузку между двумя серверами. Сразу скажу, что это не такой простой вопрос, как может показаться вначале. Со стороны nginx настройки действительно простые, но нужно будет рассмотреть отдельно вопрос общего файлового хранилища для обоих серверов и общей базы данных. Ее, скорее всего, тоже нужно будет выносить на отдельный сервер, хотя и не обязательно. Если nginx будет раздавать статику, то доступ к файлам должен быть и у него. В общем, тут нужно хорошенько все продумать и подготовить. Может как-нибудь напишу статью с примером на данную тему.

Итак, распределим нагрузку между двумя бэкендами с Apache. Вот конфигурация виртуального хоста в этом случае.

```
upstream backend {
 server 192.168.0.10:8080;
 server 192.168.0.11:8080;
}

server {
 listen 80;
 server_name example.com www.example.com;
 access_log /var/log/nginx/example.com.access.log main;
 location ~* ^.+.(js|css|png|jpg|jpeg|gif|ico|woff|woff2|swf|ttf|svg|html|txt)$ {
  root   /var/www/example.com/public;
  expires 1y;
}

 location / {
  proxy_pass http://backend/;
```

```
  proxy_set_header Host $host;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_connect_timeout    120;
  proxy_send_timeout    120;
  proxy_read_timeout    180;
 }
}
```

В данном примере балансировки нагрузки, nginx отдает всю статику, а 2 сервера с apache обрабатывают все остальные запросы.

Настройка редиректов и rewrite правил

Rewrite это то, что в первую очередь не позволяет отказаться от Apache. Многие проекты имеют массу **rewrite** правил в **.htaccess**, который поддерживает apache. Перенос этих правил в nginx не всегда прост и очевиден. Есть средства по автоматической конвертации правил rewrite из формата apache в формат nginx, но они далеко не всегда помогают. Пример такого сервиса — https://winginx.com/ru/htaccess. В идеале, такую конвертацию следует проделывать вручную, подключая голову :)

Это что касается конвертации правил из апача. А в целом rewrite правила в nginx это очень мощный инструмент. Вот несколько примеров правил.

С помощью rewrite можно отрезать у доменных имен www. Лично я считаю эту добавку к адресу сайта ненужным рудиментом и отрезаю на своих сайтах. Пример правила rewrite для замены www на запрос без него.

```
server {
 listen 443 ssl http2;
 server_name www.example.com;
 rewrite ^ https://example.com$request_uri permanent;
}
```

Это правило rewrite все запросы к домену с www переадресовывает на запрос без www. Тут это просто пример работы механизма. В данном конкретном случае, редирект www лучше сделать с помощью **return**. Это более элегантное и быстрое решение, так как не придется обрабатывать лишнюю регулярку.

На практике конкретно с www лучше поступить вот так:

```
server {
    listen 443 ssl http2;
    server_name www.example.com;
    return 301 $scheme://example.com$request_uri;
}
```

Синтаксис rewrite запросов выглядит следующим образом:

```
rewrite regex URL [flag];
```

В моем примере получаем следующие элементы правила:

- **^** — регулярное выражение;
- **https://example.com$request_uri** — url, на который заменяем;
- **permanent** — флаг, который возвращает постоянное перенаправление с кодом 301.

Вот еще один пример, который я уже приводил в разделе про location. Я настраиваю замену страниц с /amp/ на конце на обычный url без /amp/. То есть просто его обрезаю.

```
location ~ /amp/$ {
rewrite ^(.*/)amp/$ $1 permanent;
}
```

Полезным является еще одно правило rewrite, которое к ссылкам в конце без слеша добавляет слеш. То есть заменяет ссылку вида *http://example.com/page* на *http://example.com/page/*

```
rewrite ^([^.]*[^/])$ $1/ permanent;
```

Много видел в сети примеров, где rewrite используют для перенаправления запросов с http на https. Как и в случае с заменой www, так лучше не делать.

Тем более не стоит это делать через условия if, например вот так:

```
if ($scheme = "http") {
rewrite ^ https://example.com$uri permanent;
}
```

Это рабочий вариант, но в данном случае **return** вместо **rewrite** будет работать более эффективно, затрачивая меньше ресурсов web сервера. То же самое относится к примерам, где требуется замена имени домена в случае переезда. Вместо rewrite лучше использовать return. То есть делать не вот так:

```
server {
    listen 80;
    server_name old-name.com;
    rewrite ^ $scheme://new-name.com$request_uri permanent;
}
```

а так:

```
return 301 $scheme://new-name.com;
```

Такой редирект с одного домена на другой быстрее работает.

Принципиальная разница между rewrite и return в том, что в rewrite переписывается только та часть исходного URL, которая соответствует регулярному выражению, а в return весь URL-адрес переписывается на указанный URL-адрес. Из этой особенности следует то, что return работает быстрее rewrite, поэтому там, где можно использовать return, лучше использовать именно его. Условно, return следует использовать там, где требуется постоянная замена адреса, а rewrite где требуется временное изменение запроса в силу каких-то обстоятельств.

Вот пример, где без rewrite не обойтись. Допустим, у вас есть какой-то обработчик запросов, которому нужно передавать различные урлы в определенном формате. К примеру, пользователи запрашивают урл *http://example.com/linux/ubuntu*, а нам надо его преобразовать в такой — *http://example.com/linux.php?distro=ubuntu*. Делаем это с помощью следующего правила rewrite.

```
rewrite ^/linux/(.*)$ /linux.php?distro=$1 last;
```

Еще пример, как сделать постоянное перенаправление с одной страницы на другую. Как обычно, можно сделать двумя способами, с помощью rewrite или return. Более правильно использовать return. Меняем адрес *http://example.com/linux/ubuntu/* на *http://example.com/windows/win10/*

```
location = /linux/ubuntu/ {
return 301 /windows/win10/;
}
```

Вот нежелательный вариант, который тем не менее постоянно рекомендуют в разных статьях. Ошибки тут нет, но с return более правильно.

```
location = /linux/ubuntu/ {
rewrite ^/linux/ubuntu/$ /windows/win10/ permanent;
}
```

Фух, надеюсь с return и rewrite более ли менее понятно объяснил. Есть еще для перенаправлений **try_files**. Я немного плаваю в этой теме, поэтому решил не добавлять эти правила, чтобы вас не запутать и не наговорить неправды. К примеру, вопрос со слешами на конце урла в wordpress можно решить с помощью try_files таким образом:

```
try_files $uri $uri/ /index.php?$args;
```

Проверяется запрос со слешом, потом без него, если ничего не найдено, запрос уходит на index.php с параметрами. То есть не важно, что наберет пользователь, у него в любом случае будет открыта та или иная страница. Более подробно о различных правилах перенаправления можно почитать в этой англоязычной статье. Это наиболее полная информация по данной теме, что мне нагуглилась.

502 bad gateway и другие ошибки nginx

Ошибка **502 bad gateway** знакома многим пользователям интернета, не только системным администраторам. Ее рано или поздно можно увидеть на любом сайте. Что она означает? В общем случае, это значит, что на веб сервере какие-то проблемы.

Ошибка 502 описана в RFC Hypertext Transfer Protocol (HTTP/1.1) в разделе 6.6.3. Там говорится, что во время обработки запроса веб сервер, в данном случае nginx, не получил ответ от какого-то бэкенда. Расскажу, что это обычно значит на практике.

Допустим, у вас nginx работает в связке с apache или php-fpm. Вы видите описываемую ошибку. Причин может быть две:

1. Службы php-fpm или apache перестали отвечать, потому что просто упали. В таком случае, nginx будет показывать всем пользователям ошибку 502, пока бэкенд не начнет отвечать.
2. Служба просто не справляется с нагрузкой. Ошибка будет только у части пользователей, а у других все будет нормально.

То же самое может произойти, если вы настроили проксирование запросов через proxy_pass на какой-то другой сервер и этот сервер перестал отвечать. Nginx будет показывать ошибку 502 bad gateway в данном случае.

Для того, чтобы исправить 502-ю ошибку, надо разобраться, с чем конкретно она связана. Если бэкенд просто упал, то надо его поднять. Если же причина не в этом, то надо более детально разбираться. Чаще всего в логах ошибок nginx есть вся информация для решение проблем с ошибкой 502. Наиболее распространенная ситуация с этой ошибкой при работе в связке с php-fpm в том, что php-fpm просто не выдерживает нагрузки, либо неправильно

настроен. Возможно, у него не хватает процессов для обработки всех запросов. Тогда часть запросов будут возвращать ошибку.

Вот типичный пример ошибки 502 при работе с php-fpm. Пользователь видит ошибку. Идем смотреть лог ошибок виртуального хоста. Видим там такую строку:

```
2019/03/29 15:00:42 [error] 2058#2058: *18 connect() failed (111: Connection refused) while connecting to upstream, client:
192.168.13.16, server: localhost, request: "GET /1.php HTTP/1.1", upstream: "fastcgi://127.0.0.1:9000", host:
"192.168.13.34"
```

В данном случае php-fpm просто упал и перестал обрабатывать запросы.

Я достаточно часто встречаюсь с этой серверной ошибкой. Иногда разработчики напрограммируют таких конструкций, что они валят php-fpm по какой-то причине. Причем, если проект уже сдали и он давно работает, никто с этими ошибками разбираться уже не хочет. Тогда я просто ставлю костыль — автоматически перезапускаю php-fpm с помощью zabbix, ели вылезает ошибка 502 bad gateway в nginx. Такие вещи иногда годами работают и в целом всех устраивают.

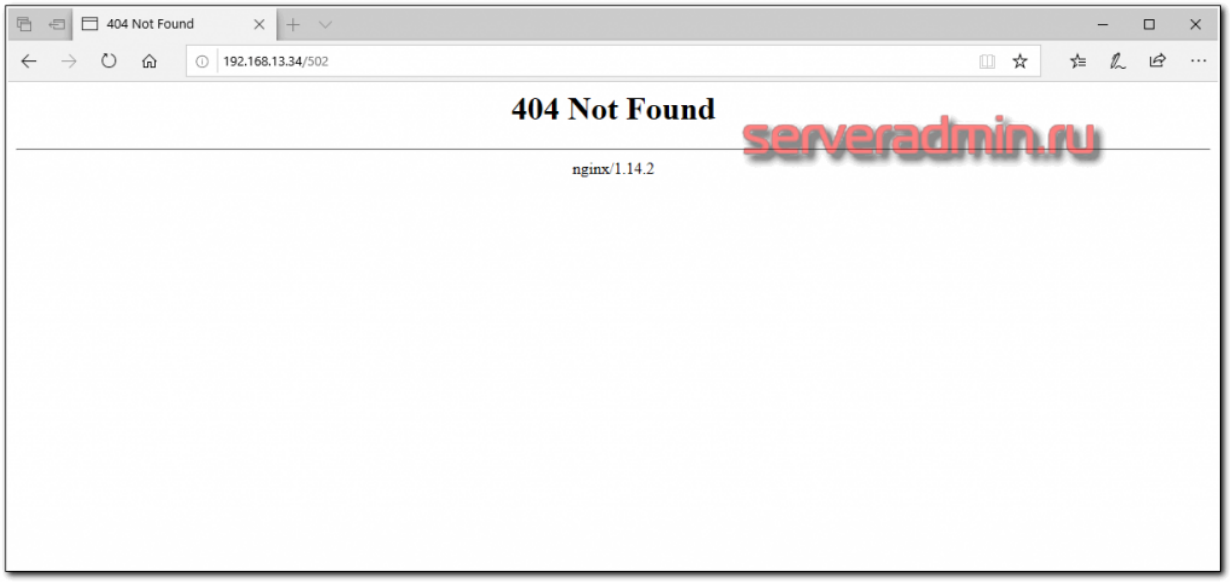
Вы можете настроить внешний вид страницы с ошибкой. То, что я показал на скрине — стандартный вид ошибки 502 bad gateway в nginx при отсутствии каких-то настроек на этот счет. Вы же можете установить свою страницу при возникновении этой ошибки. На ней можно написать, к примеру, что на сервере ведутся технические работы и скоро он заработает вновь. В каждый виртуальный хост можно установить свою страницу с ошибкой. Настраивается она в секции server.

```
error_page   500 502 503 504  /error50x.html;
 location = /50x.html {
 root   /usr/share/nginx/html;
}
```

В данном случае мы на все ошибки с кодами 500 502 503 504 показываем страницу */usr/share/nginx/html/error50x.html*.

Ошибка 404 not found

Еще одна популярная ошибка nginx — **404 not found**.

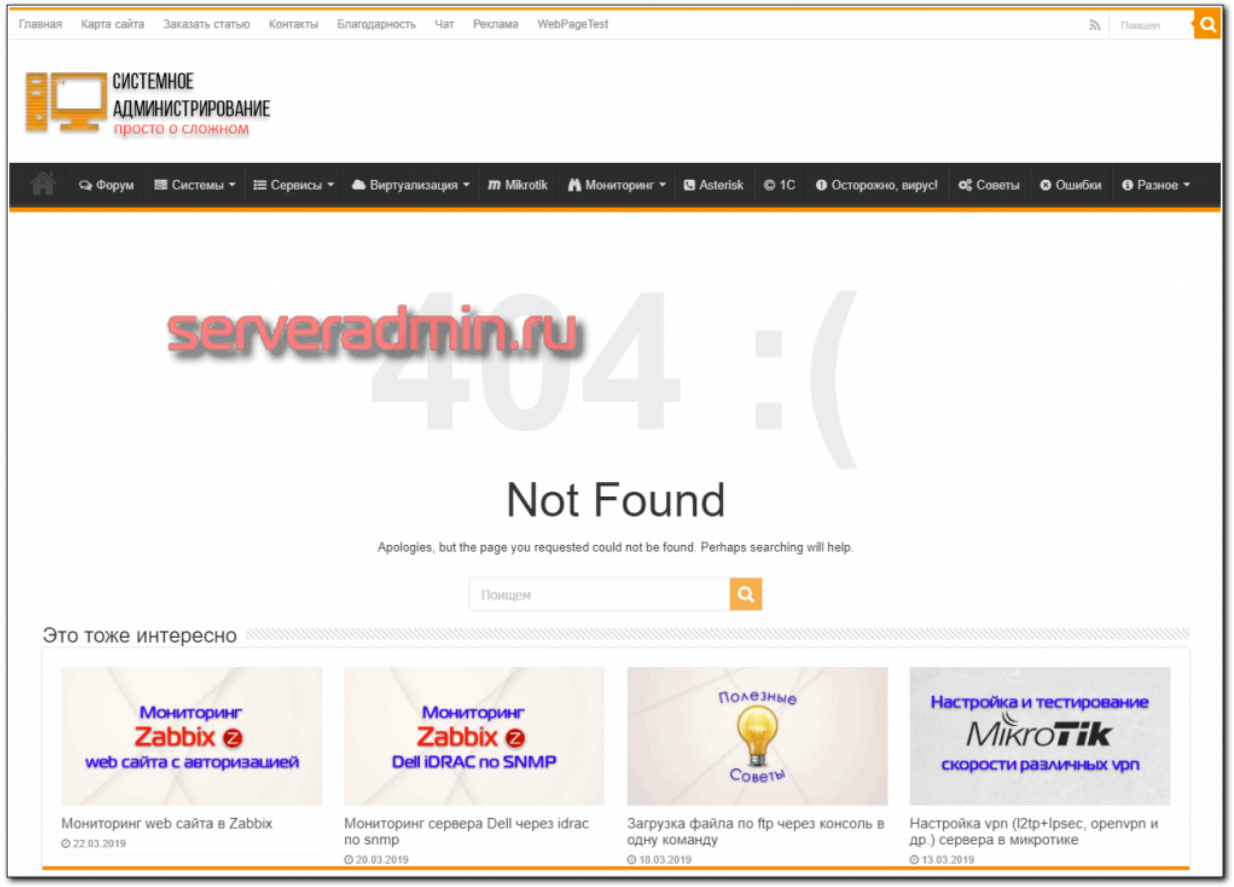


Тут все понятно и без объяснений — пользователь открывает ссылку, а документа по этой ссылке нет. Иногда бывает не очевидно, почему по ссылке выходит 404 ошибка. В таком случае рекомендую сразу смотреть лог ошибок. Вот пример.

```
2019/03/29 15:18:48 [error] 2058#2058: *20 open() "/usr/share/nginx/html/502" failed (2: No such file or directory), client: 192.168.13.16, server: localhost, request: "GET /502 HTTP/1.1", host: "192.168.13.34"
```

Ссылка ведет на документ, который должен быть в директории */usr/share/nginx/html/*, но его там нет. Когда настраиваются алиасы, часто возникают такого рода ошибки. Вроде все настроил, а все равно 404 ошибка. Идешь смотреть лог и видишь, что реально nginx ищет не там документ, где ты ожидал, согласно настройкам. Надо просто все еще раз перепроверить. В статьях про установку zabbix я регулярно вижу вопросы, связанные с 404 ошибкой. Люди просто ошибаются при настройке веб сервера.

Страницу с этой ошибкой можно так же кастомизировать. Это делают практически все крупные сайты. Вот, к примеру, как выглядит эта ошибка у меня на сайте.



В данном случае внешний вид страницы с ошибкой формирует сам wordpress. Но вы так же можете это настроить самостоятельно через nginx.

Переменные в nginx

В примерах выше с редиректами я использовал переменные, но совершенно не останавливался на них и не пояснял, что они значат. Многие из них

понятны по названиям, например:

- $args — аргументы в строке запроса;
- $request_uri — первоначальный URI запроса целиком (с аргументами);
- $scheme — схема запроса, http или https.

Полный список переменных можно посмотреть в документации. Когда вы отлаживаете сложную конфигурацию nginx с множеством переменных, бывает удобно выводить эти переменные в отдельные заголовки, а потом смотреть их во время отладки. Просто добавьте в конфигурацию виртуального хоста запись переменной примерно так.

```
add_header R-var "$request_uri";
```

Вместо R-var можете написать что угодно, это просто название записи. Далее в DevTools в Chrome можете смотреть эти заголовки.

Кэширование в nginx

Тема кэширования очень обширна. Много копий сломано о том, какое и где кэширование лучше применять. В том же wordpress существует огромное множество плагинов для кэширования. Nginx может самостоятельно хранить и управлять кэшом. В некоторых случаях это будет эффективнее, чем использовать плагины. Но все сильно зависит от конкретного проекта.

Кэшировать Nginx может разные вещи:

- Статику, которую получает с удаленного сервера, сохраняет себе и раздает быстрее, чем удаленный сервер.
- Динамику, превращая ее в статику и раздавая самостоятельно, без обращения к бэкенду.

С кэширвоанием статики все более ли менее понятно. Сохраняем файлы и отдаем их сами. А вот с кэшированием динамических страниц есть очень много нюансов. Конечно, удобно сформировать один раз динамическую страницу php, сохранить ее как html и раздавать. А что делать со счетчиком просмотров на странице, с комментариями, со списком последних статей, после публикации новой? Это первое, что приходит в голову. Таких моментов может быть очень много. В реальном проекте скорее всего не получится просто взять и включить кэширование.

Конкретно с WordPress я обычно поступаю следующим образом. Кэширование nginx я не использую. Вместо этого я использую плагин WP Total Cache. Он формирует статические html страницы по заданным в его настройках параметрам. А дальше я эти страницы отдаю напрямую через nginx, минуя вообще ядро WordPress. За счет этого достигается максимальное быстродействие. Вот пример настроек Nginx из секции server виртуального хоста для отдачи

кэша WordPress.

```
set $cache_uri $request_uri;
if ($request_method = POST) {
 set $cache_uri 'null cache';
}
if ($query_string != "") {
 set $cache_uri 'null cache';
}
if ($request_uri ~* "(/wp-admin/|/xmlrpc.php|/wp-(app|cron|login|register|mail).php|wp-.*.php|/feed/|index.php|wp-comments-
popup.php|wp-links-opml.php|wp-locations.php|sitemap(_index)?.xml|[a-z0-9_-]+-sitemap([0-9]+)?.xml)") {
 set $cache_uri 'null cache';
}
if ($http_cookie ~* "comment_author|wordpress_[a-f0-9]+|wp-postpass|wordpress_logged_in") {
 set $cache_uri 'null cache';
}
location / {
 try_files /wp-content/cache/supercache/$http_host/$cache_uri/index-https.html $uri $uri/ /index.php?$args;
}
```

Сначала идут проверки для добавления исключений к некоторым запросам, для которых кэширование не будет работать. А потом отдается статика из директории с кэшом. Если для заданного URI кэша нет, он уходит дальше в обработку.

В данном случае используется именно плагин WP, а не кэш nginx только из-за удобства управления кэшом через панель управления сайтом. Сам плагин делает ровно то же самое, что может делать nginx. Кэш в самом nginx настраивается следующим образом.

Сначала добавляются настройки кэширования в *nginx.conf*. Дальше речь пойдет о кэшировании динамики через fastcgi.

```
http {
...
 fastcgi_cache_path /var/cache/nginx/php levels=1:2 keys_zone=php_cache:32m max_size=3g;
 fastcgi_cache_key "$scheme$request_method$host$request_uri";
```

```
...
}
```

- fastcgi_cache_path — директива для объявления кэша для fatcgi;
- /var/cache/nginx/php — директория, где будут храниться файлы кэша;
- levels=1:2 — уровень вложенности каталогов в директории с кэшом;
- keys_zone=php_cache — название зоны;
- max_size=3g — размер директории с кэшом в 3Гб.

Не забудьте создать указанную директорию для кэша */var/cache/nginx/php*. В конфигурацию виртуального хоста добавляем параметры кэширования в location с php.

```
location ~ \.php$ {
 fastcgi_pass unix:/var/run/php-fpm.sock;
 fastcgi_index index.php;
 include fastcgi_params;
 fastcgi_cache php_cache;
 fastcgi_cache_valid 200 120m;
 }
}
```

Кэшируем ответы с кодом 200 на 120 минут.

Для кэширования запросов с ответами от бэкенда через proxy_pass, необходимо использовать директиву **proxy_cache_path**.

```
proxy_cache_path /var/cache/nginx/proxy levels=1:2 keys_zone=proxy_cache:32m max_size=3G;
```

И далее в вирутальном хосте.

```
location / {
 proxy_pass http://backend;
 proxy_cache proxy_cache;
```

```
 proxy_cache_valid 200 120m;
}
```

Подробнее о кэшировании читайте в блоге nginx. Там очень много нюасов. Как минимум надо аккуратно прорабатывать исключения, чтобы в кэш не попадало то, что там быть не должно. Например, cookie или страницы из закрытой административной части.

Auth basic, доступ по паролю или ограничение по ip

Покажу на простых примерах, как в nginx настроить ограничения доступа по ip, имени пользователю и паролю (Auth basic авторизация). Начнем с простого. Закроем доступ к определенной папке на сайте всем подряд и разрешим только после ввода имени пользователя и пароля. Для этого добавляем в виртуальный хост, в нужный location следующие параметры.

```
location /secret {
...
 auth_basic "Unauthorized";
 auth_basic_user_file /etc/nginx/htpasswd;
...
}
```

Теперь нам надо создать файл с именем пользователя и паролем.

```
htpasswd -c /etc/nginx/htpasswd user
New password:
Re-type new password:
Adding password for user user
```

Если получите сообщение, что у вас нет утилиты **htpasswd**, установите соответствующий пакет.

```
# apt install apache2-utils
# yum install httpd-tools
```

Перечитайте конфигурацию nginx и проверяйте. Теперь доступ к /secret возможен только после авторизации по имени пользователю и паролю.

Настроим ограничение доступа по ip в nginx. Для этого достаточно добавить в свойства location или всего виртуального хоста следующие правила доступа.

```
location /secret {
...
 allow 192.168.10.10/32
 deny all;
...
}
```

Если нужен запрет доступа ко всему сайту сразу, то ставьте это ограничение в секцию server.

Если вам нужно настроить ограничение доступа по ip на основе стран или регионов, то читайте мою отдельную статью на эту тему — блокировка доступа к сайту по странам.

Мониторинг nginx

Для настройки мониторинга nginx, необходимо внести некоторые параметры в конфигурационный файл *nginx.conf*. После этого nginx сам будет отдавать базовую информацию о состоянии сервера с помощью модуля **ngx_http_stub_status_module**. Добавляем в секцию http следующее.

```
server {
 listen localhost;
 server_name status.localhost;
 keepalive_timeout 0;
 allow 127.0.0.1;
 deny all;
 location /server-status {
  stub_status on;
 }
access_log off;
}
```

Перезапускаем nginx и проверяем. Я разрешил отдавать состояние о своем статусе только при запросе с локального сервера, где сам nginx работает. Поэтому смотрим через консоль сервера.

```
# curl http://localhost/server-status
```

Какие метрики мы здесь видим:

- Active connections — количество активных клиентских соединений.
- accepts — число принятых клиентских соединений.
- handled — число обработанных соединений.
- requests — число клиентских запросов.
- Reading — число соединений, в которых nginx в настоящий момент читает заголовок запроса.
- Writing — число соединений, в которых nginx в настоящий момент отвечает клиенту.
- Waiting — число бездействующих клиентских соединений в ожидании запроса.

Что делать с этими данными — решать вам в зависимости от того, какую систему мониторинга вы используете. Более подробно о мониторинге nginx читайте в отдельной статье.

Пример универсального конфига для nginx

В завершении своей статьи про настройку nginx, я хочу привести шаблон универсального конфига, который я обычно использую при настройке веб сервера. Сил уже нет писать статью, поэтому привожу его без подробных комментариев. Надеюсь сами разберетесь с помощью документации. Статью писал несколько дней и под конец уже устал :)

```
user  nginx;
worker_processes auto;
worker_cpu_affinity auto;
worker_rlimit_nofile 30000;
pid /var/run/nginx.pid;
```

```
pcre_jit on;

events {
    worker_connections 8192;
    multi_accept on;
}

http {

    # Basic ######################
    sendfile                    on;
    tcp_nopush                  on;
    tcp_nodelay                 on;
    reset_timedout_connection   on;
    keepalive_timeout           120;
    keepalive_requests          1000;
    types_hash_max_size         2048;
    server_tokens               off;
    send_timeout                30;
    client_body_timeout         30;
    client_header_timeout       30;
    server_names_hash_max_size  4096;

    # Limits #####################
    client_max_body_size        10m;
    client_body_buffer_size     128k;
    client_body_temp_path       /var/cache/nginx/client_temp;

    proxy_connect_timeout       5;
    proxy_send_timeout          10;
    proxy_read_timeout          10;
    proxy_buffer_size           4k;
    proxy_buffers               8 16k;
```

```
    proxy_busy_buffers_size     64k;
    proxy_temp_file_write_size  64k;
    proxy_temp_path             /var/cache/nginx/proxy_temp;

    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    # Logs ########################

    log_format  main    '$remote_addr - $host [$time_local] "$request" '
            '$status $body_bytes_sent "$http_referer" '
            '"$http_user_agent" "$http_x_forwarded_for"'
            'rt=$request_time ut=$upstream_response_time '
            'cs=$upstream_cache_status';
    log_format full     '$remote_addr - $host [$time_local] "$request" '
               'request_length=$request_length '
               'status=$status bytes_sent=$bytes_sent '
               'body_bytes_sent=$body_bytes_sent '
               'referer=$http_referer '
               'user_agent="$http_user_agent" '
               'upstream_status=$upstream_status '
               'request_time=$request_time '
               'upstream_response_time=$upstream_response_time '
               'upstream_connect_time=$upstream_connect_time '
               'upstream_header_time=$upstream_header_time';

    access_log  /var/log/nginx/access.log  main;
    error_log  /var/log/nginx/error.log;

    # Gzip ########################

    gzip on;
    gzip_static on;
```

```
    gzip_types text/plain text/css application/json application/x-javascript text/xml application/xml application/xml+rss
text/javascript application/javascript image/x-icon image/svg+xml application/x-font-ttf;
    gzip_comp_level 9;
    gzip_proxied any;
    gzip_min_length 1000;
    gzip_disable "msie6";
    gzip_vary on;

    etag off;
    # Если собран с модулем сжатия brotli
    #brotli_static on;
    #brotli on;
    #brotli_comp_level 6;
    #brotli_types text/plain text/css text/xml application/javascript image/x-icon image/svg+xml;

    # Cache ########################

    #proxy_cache_valid 1m;
    #proxy_cache_key $scheme$proxy_host$request_uri$cookie_US;
    #proxy_cache_path /web/sites/nginx_cache levels=1:2 keys_zone=main:1000m;

    # Zone limits ################

    limit_conn_zone $binary_remote_addr zone=perip:10m;
    limit_req_zone $binary_remote_addr zone=lim_5r:10m rate=5r/s; # lim for dynamic page
    limit_req_zone $binary_remote_addr zone=lim_1r:10m rate=1r/s; # lim for search page
    limit_req_zone $binary_remote_addr zone=lim_10r:10m rate=10r/s;

    # SSL ##########################

    ssl_session_cache shared:SSL:50m;
    ssl_session_timeout 1d;
    ssl_session_tickets on;
```

```
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;
    ssl_ciphers 'TLS13-CHACHA20-POLY1305-SHA256:TLS13-AES-128-GCM-SHA256:TLS13-AES-256-GCM-SHA384:ECDHE:ECDHE-ECDSA-
CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-
RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-
AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-
ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-
SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS';
    ssl_prefer_server_ciphers on;
    ssl_dhparam /etc/ssl/certs/dhparam.pem;
    ssl_stapling on;
    ssl_stapling_verify on;
    add_header Strict-Transport-Security max-age=15768000;
    resolver 8.8.8.8;

    include /etc/nginx/conf.d/*.conf;

    # For monitoring ############
    server {
    listen localhost;
    server_name status.localhost;
    keepalive_timeout   0;
    allow       127.0.0.1;
    deny        all;
    access_log  off;

        location /server-status {
        stub_status        on;
        }

        location /status {
        access_log off;
        allow 127.0.0.1;
```

```
        deny all;
        include fastcgi_params;
        fastcgi_pass   unix:/var/run/php-fpm/php-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        }
    }
}
```

На этом по базовой настройке nginx все. Надеюсь, было полезно.

Заключение

> Не понравилась статья и хочешь научить меня администрировать? Пожалуйста, я люблю учиться. Комментарии в твоем распоряжении. Расскажи, как сделать правильно!

Я постарался собрать в обзорной статье все то, с чем приходится обычно работать в Nginx. В целом, этот продукт хорошо документирован, популярен. В интернете есть много полезной информации по всем темам в отдельности. Но похожей объемной статьи по продукту я не нашел, поэтому решил написать.

Остались несколько тем, которые я хотел бы тоже расписать, но не успел. Да и статья получилась очень большая. Не знаю, насколько это уместно, публиковать все в куче. В общем, не рассмотрены остались вопросы логирования, gzip сжатия и лимитов. Тема логирования частично рассмотрена в статье про мониторинг бэкенда с помощью elk.

Так же рекомендую в продолжении темы мониторинга nginx посмотреть статью про дашборды в kibana для nginx. Полезный и удобный инструмент для администраторов сайтов и web серверов.

Буду рад замечаниям, исправлениям, дополнениям в комментариях к статье.