Nginx. Установка и настройка

13.12.2023

Теги: CLI · Nginx · Конфигурация · Настройка · Сервер · Установка

Nginx – это веб-сервер с открытым исходным кодом, созданный работать под высокой нагрузкой, чаще всего используемый для отдачи статического контента. Он также может выполнять другие важные функции, такие как балансировка нагрузки, кеширование HTTP и использование в качестве обратного прокси.

Установка nginx-core

Обновляем списки пакетов из репозиториев и устанавливаем

```
$ sudo apt update
$ sudo apt install nginx
```

Копировать

После установки добавляем службу в автозагрузку

```
$ sudo systemctl enable nginx.service
```

Копировать

Проверяем статус работы веб-сервера

```
$ sudo systemctl status nginx.service
```

Копировать

Проверяем наличие службы в автозагрузке

```
$ sudo systemctl is-enabled nginx.service
```

Копировать

Для работы с установленным веб-сервером пригодятся базовые команды управления

Функция	Команда
Запуск	sudo systemctl start nginx
Остановка	sudo systemctl stop nginx
Перезапуск	sudo systemctl restart nginx
Перезагрузка	sudo systemctl reload nginx
Проверка состояния	sudo systemctl status nginx
Тест конфигурации	sudo nginx -t

Варианты установки

Категории блога

Web-разработка
HTML и CSS
JavaScript и TypeScript
PHP и MySQL
CMS Битрикс
CMS WoprdPress
Yii2 и Laravel
Python и Flask
Web-аналитика
Разное
ОС Linux
Команды
Сценарии
Разное
1С:Предприятие
Программирование
Язык запросов
Разное
Локальная сеть
Разное

Облако тегов

1С:Предприятие (31)
API (29)
Bash (43)
CLI (124)
CMS (139)
CSS (50)
Frontend (75)
HTML (66)
JavaScript (150)
Laravel (72)
Linux (171)
MySQL (76)
PHP (125)
React.js (66)
SSH (27)
Ubuntu (69)
Web-разработка (509)
WordPress (73)
Yii2 (69)
БазаДанных (95)
Битрикс (66)
Блог (29)
Верстка (43)
ИнтернетМагаз··· (84)
КаталогТоваров (87)
Класс (30)
Клиент (28)
Ключ (28)
Команда (88)
Компонент (60)
Конфигурация (66)
Корзина (32)
ЛокальнаяСеть (32)
Модуль (34)
Навигация (31)
Настройка (143)

Для установки доступны пакеты nginx-light, nginx-core (наш вариант), nginx-full и nginx-extra.

Пакет nginx-light – это облегченная версия nginx, с минимальным набором модулей. Включает в себя только базовые возможности HTTP-сервера, недоступны многие модули, например модуль перезаписи URL ngx_http_rewrite_module или модуль сжатия ngx_http_gzip_module. Может быть полезен в контейнерных средах, таких как Docker, где минимализм является ключевым моментом. Это также неплохой выбор для основных задач балансировки нагрузки.

Пакет nginx-core представляет собой минимальную, но полную установку веб-сервера nginx. Это отправная точка, если нужен функциональный сервер без посторонних модулей. Включает в себя базовые функции HTTP, такие как поддержка HTTP/2, поддержка Secure Sockets Layer (SSL) и основные функции обратного прокси. Хороший выбор для простых веб-приложений, статических веб-сайтов или конфигураций обратного прокси-сервера, где не нужны дополнительные функции.

Пакет nginx-full – более многофункциональная версия nginx, предназначенная для более сложных и ресурсоемких задач. Включает в себя почти все модули с исходным исходным кодом nginx, такие как возможности проксирования, дополнительные модули HTTP, поддержку WebSocket и многое другое. Подходит для сложных веб-приложений, которым требуются различные функции, такие как SSL, проксирование и расширенные возможности кэширования. Также хорошо подходит, если нужен веб-сервер общего назначения, не слишком беспокоясь о том, какие модули могут понадобиться позже.

Пакет nginx-extras – наиболее многофункциональный, ориентирован на узкоспециализированные развертывания, требующие широкого набора функциональных возможностей. Включает в себя все модули, имеющиеся в nginx-full, а также дополнительные сторонние модули. Однако следует отметить, что nginx-extras не включает в себя все возможные модули. Для тех нишевых потребностей, которые не удовлетворяются доступными пакетами, может потребоваться компиляция nginx из исходного кода.

Файлы конфигурации

Все файлы конфигурации Nginx расположены в директории /etc/nginx

Копировать

```
$ ls -la /etc/nginx/
total 72
drwxr-xr-x  8 root root 4096 дек 11 15:43 .
drwxr-xr-x 99 root root 4096 дек 11 15:44 ..
drwxr-xr-x  2 root root 4096 мая 30  2023 conf.d
-rw-r--r--  1 root root 1125 мая 30  2023 fastcgi.conf
-rw-r--r--  1 root root 1055 мая 30  2023 fastcgi_params
-rw-r--r--  1 root root 2837 мая 30  2023 koi-utf
-rw-r--r--  1 root root 2223 мая 30  2023 koi-win
-rw-r--r--  1 root root 3957 мая 30  2023 mime.types
drwxr-xr-x  2 root root 4096 мая 30  2023 modules-available
drwxr-xr-x  2 root root 4096 дек 11 15:44 modules-enabled
-rw-r--r--  1 root root 1447 мая 30  2023 nginx.conf
-rw-r--r--  1 root root  180 мая 30  2023 proxy_params
-rw-r--r--  1 root root  636 мая 30  2023 scgi_params
```

ПанельУправле… (29) Установка (67)
Плагин (33) Файл (51)
Пользователь (26) Форма (58)
Практика (101) Фреймворк (192)
Сервер (77) Функция (36)
Событие (28) ШаблонСайта (68)
Теория (106)

Все теги

Категории статей

Web-разработка
1С:Предприятие
ОС Linux
Регулярные выражения
Разное

```
drwxr-xr-x  2 root root 4096 дек 11 15:43 sites-available
drwxr-xr-x  2 root root 4096 дек 11 15:43 sites-enabled
drwxr-xr-x  2 root root 4096 дек 11 15:43 snippets
-rw-r--r--  1 root root  664 мая 30  2023 uwsgi_params
-rw-r--r--  1 root root 3071 мая 30  2023 win-utf
```

Здесь много чего, но нам в первую очередь интересны следующие файлы и директории

- nginx.conf – главный конфигурационный файл nginx.
- conf.d – дополнительные конфигурационные файлы nginx.
- sites-available – в этой директории хранятся файлы виртуальных хостов. Nginx не использует файлы из этой директории, если ссылки на них нет в директории sites-enabled.
- sites-enabled – директория, в которой хранятся активированные виртуальные хосты. Обычно это делается путём создания ссылки на файл конфигурации хоста из директории sites-available.
- modules-available, modules-enabled – эти директории содержат, соответственно, доступные и активные модули.

Главный файл конфигурации

Давайте посмотрим содержимое главного файла конфигурации /etc/nginx/nginx.conf

```
    keepalive_timeout 40;

    # настройки кэширования
    open_file_cache max=10000 inactive=30s;

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

    # настройки SSL
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping
    ssl_prefer_server_ciphers on;

    # логирование
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    # разрешить сжатие
    gzip on;

    # виртуальные хосты
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

Файл конфигурации состоит из секций и директив

```
секция main
 ├─── директивы main
 ├─── секция events
```

```
│     └──── директивы events
├──── секция http
│     ├──── директивы http
│     ├──── секция server
│     │     ├──── директивы server
│     │     └──── секция location
│     └──── секция server
│           ├──── директивы server
│           └──── секция location
└──── секция mail
      ├──── директивы mail
      ├──── секция server
      │     └──── директивы server
      └──── секция server
            └──── директивы server
```

Модульная структура

Nginx является модульным приложением. Это значит, что каждая из директив в конфиге связана с определенным модулем. Так, к примеру, модуль ngx_core_module отвечает за основную логику – то есть определяет как раз те директивы, без которых сервер просто не запустится (например user).

Модули существуют двух видов – статические и динамические. Статические – это модули, которые входят в состав nginx в результате сборки из исходников с добавлением кода необходимого модуля. Динамические – это модули, которые можно подключать к nginx без сборки самого nginx, хотя сам модуль должен быть собран как внешняя библиотека. Однако nginx не позволяет подключить динамический модуль, если в ходе сборки nginx не была указана в параметрах сборки поддержка этого модуля. Иначе говоря – если требуется конкретный модуль, то в любом случае необходимо собирать nginx из исходных файлов. А вот как – уже на ваше усмотрение и исходя из возможностей работы модуля (не все модули умеют работать как динамические).

В директории /etc/nginx/modules-enabled мы найдем несколько символических ссылок на файлы конфигурации. В этих файлах конфигурации с помощью директивы load_module загружаются динамические модули.

Копировать
```
$ ls -la /etc/nginx/modules-enabled
lrwxrwxrwx 1 root root   55 дек 11 15:44 50-mod-http-geoip2.co
lrwxrwxrwx 1 root root   61 дек 11 15:44 50-mod-http-image-fil
lrwxrwxrwx 1 root root   60 дек 11 15:44 50-mod-http-xslt-filt
lrwxrwxrwx 1 root root   48 дек 11 15:44 50-mod-mail.conf -> /
lrwxrwxrwx 1 root root   50 дек 11 15:44 50-mod-stream.conf ->
lrwxrwxrwx 1 root root   57 дек 11 15:44 70-mod-stream-geoip2.
```

Копировать
```
$ cat /etc/nginx/modules-enabled/50-mod-http-image-filter.conf
load_module modules/ngx_http_image_filter_module.so;
```

Модуль ngx_http_image_filter_module позволяет использовать в файле конфигурации директиву image_filter, с помощью которой можно изменять размер, поворачивать и обрезать изображения. Модуль ngx_http_empty_gif_module позволяет использовать директиву empty_gif, которая выдает прозрачное gif-изображение размером 1x1 пиксель.

```
location /img/ {
    proxy_pass   http://backend;
    image_filter resize 150 100;
    image_filter rotate 90;
    error_page   415 = /empty;
}

location = /empty {
    empty_gif;
}
```

Директивы конфигурации

Существует два вида директив – простые и блочные. Простая директива состоит из имени и параметров, разделённых пробелами, и в конце строки ставится точкой с запятой. Блочная директива (секция) состоит из нескольких директив, помещённых внутри фигурных скобок.

- user – пользователь, от имени которого работает nginx, обычно www-data
- worker_processes - количество процессов сервера, значение выставляется равным количеству ядер процессора, auto – nginx определит автоматически
- pid – файл, внутри которого хранится идентификатор запущенного главного процесса (PID)
- include – подключаемый файл или файлы конфигурации
- events – секция директив, определяющих работу с сетевыми соединениями
- worker_connections – по умолчанию 512. Устанавливает максимальное количество соединений одного рабочего процесса, то есть nginx будет обрабатывать worker_processes * worker_connections, остальные запросы ставить в очередь. Следует выбирать значения от 1024 до 4096 – зависит от производительности процессора.
- multi_accept – по умолчанию off. Если включен, позволяет принимать максимально возможное количество соединений. Если отключен, nginx решает, какой дочерний процесс будет обрабатывать запрос. Поскольку nginx очень эффективен в этом – значение off подойдет в большинстве случаев.
- http – секция директив http-сервера
- sendfile – позволяет использовать более совершенный системный вызов, который обеспечивает прямую передачу файла, то есть без системных вызовов read и write
- tcp_nopush – позволяет передавать заголовок ответа и начало файла в одном пакете, при использовании значения on – повышает производительность.
- types_hash_max_size – регламентирует максимальный размер хэш таблиц типов. Чтобы быстро обрабатывать статические наборы данных – nginx использует хеш-таблицы. При запуске nginx сам подбирает размеры хеш-таблиц – но не больше указанного значения. Чем больше значение, тем больше

памяти будет использовано, но частота конфликтов хеш-ключей уменьшится, а скорость извлечения будет выше. Если в логах появляются сообщения «could not build optimal types_hash» – нужно увеличить значение.

- keepalive_timeout – по умолчанию 75. Отвечает за максимальное время поддержания keepalive-соединения, в случае, если клиент по нему ничего не запрашивает. Для современных условий, стоит выставить от 30 до 50.
- open_file_cache – по умолчанию отключена. При включении nginx будет хранить в кэше дескрипторы открытых файлов, информацию об их размерах и времени модификации, информацию о существовании каталогов, информацию об ошибках поиска файла – «файл не найден», «нет прав на чтение».
- default_type – указывает тип MIME ответа по умолчанию
- ssl_protocols – включает указанные SSL протоколы
- access_log – задает путь к файлу лога доступа и формат записей; при выставлении значения в off, запись в журнал доступа будет отключена
- error_log – путь к файлу лога регистрации ошибок и уровень детализации
- gzip – включает или отключает сжатие

Логирование запросов и ошибок

Директива log_format позволяет описать, что будет содержаться в журнале запросов. По умолчанию используется формат combined.

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

Все, что начинается с доллара – это переменные, большинство из которых доступно по умолчанию. Однако, возможно задать новые переменные, которые также можно использовать при описании формата логов доступа.

Лог ошибок, как нетрудно догадаться, используется для записи ошибок работы nginx. В отличие от access_log, в директиву error_log передается не формат логов, а минимальный уровень детализации. По умолчанию используется уровень error, но существуют и другие уровни, вплоть до debug.

```
error_log /var/log/nginx/error.log debug
```

Для повышения производительности при записи логов доступа можно использовать буфер перед записью на диск. Есть два тригера для записи из буфера в файл, которые можно использовать одновременно

- если буфер заполнен и больше не вмещается
- если данные в буфере старше, чем указано

Размер буфера и время сброса на диск задаются с помощью директив buffer и flush

```
access_log /var/log/nginx/access.log combined buffer=256k flus
error_log /var/log/nginx/error.log warn;
```

Виртуальные хосты

Директива server определяет так называемый виртуальный хост, что позволяет на одном ip-адресе обрабатывать несколько доменов. К какому домену идет обращение – nginx определяет по http-заголовку Host. В общем случае, каждый server может быть запущен на конкретном ip-адресе и порту, либо обрабатывать конкретное (или несколько конкретных) доменных имен.

В файле конфигурации /etc/nginx/nginx.conf есть две директивы include

```
http {
    # виртуальные хосты
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

Давайте посмотрим содержимое директории /etc/nginx/sites-enabled

```
$ ls -la /etc/nginx/sites-enabled
total 8
drwxr-xr-x 2 root root 4096 дек 11 15:43 .
drwxr-xr-x 8 root root 4096 дек 11 17:21 ..
lrwxrwxrwx 1 root root   34 дек 11 15:43 default -> /etc/nginx
```

Давайте посмотрим содержимое файла /etc/nginx/sites-enabled/default

```
# Default server configuration
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;

    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;

    server_name _;

    location / {
            # First attempt to serve request as file, then as
            # directory, then fall back to displaying a 404.
            try_files $uri $uri/ =404;
    }
}
```

Это конфигурация так называемого сервера по умолчанию. Если пришел http-запрос, но ни один из виртуальных хостов его не обработал – тогда запрос будет обработан сервером по

умолчанию. Сервер по умолчанию можно задать явно с помощью параметра default_server в директиве listen. В противном случае сервером по умолчанию будет считаться первый сервер в списке.

```
server {
    listen 80; # сервер по умолчанию для 80-го порта
    server_name example.org www.example.org;
    # ...прочие директивы конфигурации...
}

server {
    listen 80;
    server_name example.net www.example.net;
    # ...прочие директивы конфигурации...
}

server {
    listen 80;
    server_name example.com www.example.com;
    # ...прочие директивы конфигурации...
}
```

Сервер по умолчанию является свойством слушающего порта, поэтому у разных портов могут быть определены разные серверы по умолчанию

```
server {
    listen 80;
    listen 8080 default_server;
    server_name example.org www.example.org;
    # ...прочие директивы конфигурации...
}

server {
    listen 80 default_server;
    listen 8080;
    server_name example.com www.example.com;
    # ...прочие директивы конфигурации...
}
```

Директивы listen и server_name

Директива listen позволяет определить, где слушает конкретный сервер – какой порт, какой ip-адрес или вообще unix-сокет. Кроме этого, он позволяет установить большое количество опций по работе сервера, например, использовать ли SSL, по какому протоколу работать и так далее.

Директива server_name позволяет указать, какие доменные имена будет обрабатывать конкретный сервер. Можно указать несколько доменных имен через пробел. Кроме того, допускается использовать * – например, чтобы обрабатывать все субдомены *.example.com

```
server {
    listen 443 http2 ssl;
    server_name example.com www.example.com;
    # ...прочие директивы конфигурации...
}
```

Когда приходит http-запрос – nginx вначале сопоставляет ip-адрес и порт запроса с директивами listen в секциях server. Затем сопоставляет значение поле Host заголовка запроса с директивами server_name в секциях server, которые соответствуют ip-адресу и порту. Если имя сервера не найдено, запрос будет обработан сервером по умолчанию.

Давайте рассмотрим вот такой пример конфигурации

Копировать

```
server {
    listen  192.168.1.1:80;
    server_name example.org www.example.org;
    # ...прочие директивы конфигурации...
}

server {
    listen 192.168.1.1:80 default_server;
    server_name example.net www.example.net;
    # ...прочие директивы конфигурации...
}

server {
    listen 192.168.1.2:80 default_server;
    server_name example.com www.example.com;
    # ...прочие директивы конфигурации...
}
```

Запрос www.example.com, который пришел на 192.168.1.1:80, будет обработан сервером по умолчанию (второй в списке), потому как имени www.example.com нет в директиве server_name первого сервера и нет в директиве server_name второго сервера. Это значит, что запрос должен обработать сервер по умолчанию. Если бы не было параметра default_server для второго сервера – запрос бы обработал первый сервер.

Если необходимо обрабатывать http-запросы без поля Host в заголовке – нужно указать в качестве значения директивы server_name пустую строку. Это можно сделать для уже существующей секции, так и создать новую секцию server. Если это новая секция server – то директиву server_name с пустой строкой можно не указывать, это значение по умолчанию.

Копировать

```
server {
    listen 80;
    server_name example.org www.example.org "";
    # ...прочие директивы конфигурации...
}
```

Копировать

```
server {
    listen 80;
    server_name "";
    # ...прочие директивы конфигурации...
}
```

Если в запросе вместо имени сервера указан ip-адрес, то поле Host заголовка запроса будет содержать ip-адрес, и запрос можно обработать, используя ip-адрес как имя сервера.

Копировать

```
server {
    listen 80;
```

```
    server_name example.org www.example.org "" 192.168.1.1;
    # ...прочие директивы конфигурации...
}
```

Если запросы без поля Host в заголовке не должны обрабатываться, можно определить сервер, который будет их отклонять. В примере ниже возвращается специальный для nginx код 444, который закрывает соединение.

Копировать

```
server {
    listen 80;
    server_name "";
    return 444;
}
```

В примерах конфигурации серверов, обрабатывающих все запросы, встречается странное имя _ (подчеркивание). Оно не является каким-то особенным, это просто одно из множества некорректных доменных имён, которые никогда не пересекутся ни с одним из реальных имён.

Копировать

```
server {
    listen 80 default_server;
    server_name _;
    return 444;
}
```

Директивы location и root

Директивы location служит для установки конфигурации в зависимости от URI-запроса. Директива root задает начальную часть пути к файлам, которые будет отдавать nginx, в файловой системе сервера.

Копировать

```
http {
    server {
        listen 80;
        server_name example.org;

        root /var/www/example.org;

        location /img {
            # ...прочие директивы конфигурации...
        }

        location /pdf {
            # ...прочие директивы конфигурации...
        }
    }
}
```

Здесь все просто – nginx сравнивает строку, которая идет после директивы location с URI http-запроса. Скажем, приходит http-запрос http://example.org/img/logo.png, URI этого запроса – это /img/logo.png. Сравнивается начальная часть URI /img/logo.png со строкой /img – здесь есть совпадение /img с /img. Это значит, что nginx будет искать в файловой системе файл /var/www/example.org/img/logo.png – это root + /img/logo.png.

Директива location может быть представлена одним из трех вариантов

```
# поиск точного совпадения URI http-запроса со строкой string,
location = string {
    # ...прочие директивы конфигурации...
}
```

```
# поиск совпадения начала URI http-запроса со строкой string м
location [^~] string {
    # ...прочие директивы конфигурации...
}
```

```
# поиск первого совпадения URI http-запроса с регулярным выраж
location ~[*] regexp {
    # ...прочие директивы конфигурации...
}
```

Nginx ищет подходящий блок location по следующему алгоритму

1. Поиск точного совпадения URI http-запроса со строкой location = string {...} без использования регулярных выражений. Если совпадение найдено – поиск завершается.

2. Поиск совпадения начала URI http-запроса со строкой location [^~] string {...} без использования регулярных выражений. Идет поиск совпадения максимальной длины – это значит, что поиск не завершается после первого совпадения. Если совпадение максимальной длины имеет префикс ^~ – поиск завершается. Если совпадение максимальной длины не имеет префикса – location временно сохраняется.

3. Поиск совпадения URI http-запроса с шаблоном location ~ [*] regexp {...}, в порядке их определения в файле конфигурации. Если совпадение найдено – поиск завершается. Обратите внимание, что поиск завершается после первого найденного совпадения. Звездочка * означает поиск без учета регистра.

4. Возвращается временно сохраненный location из втрого пункта.

Обратите внимание, что этот алгоритм не применим при наличии вложенных location.

Примеры location и root

Давайте рассмотрим пример файла конфигурации и посмотрим, как nginx будет искать подходящую директиву location. Для этого создадим файл конфигурации example.com в директории /etc/nginx/sites-available. И поставим символическую ссылку на него в директории /etc/nginx/sites-enabled. Кроме того, нам нужна директория /var/www/example.com и внутри нее – несколько директорий с файлами изображений.

```
$ sudo mkdir /var/www/example.com
$ sudo cp /etc/nginx/sites-available/default /etc/nginx/sites-
$ sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx
```

Структура директорий внутри /var/www/example.com

```
[example.com]
    [111]
        [img]
            nginx.png
            [logo]
                nginx.png
    [222]
        [img]
            nginx.png
            [logo]
                nginx.png
    [333]
        [img]
            nginx.png
            [logo]
                nginx.png
    [444]
        [img]
            nginx.png
            [logo]
                nginx.png
```

Смысл этого в том, что файл nginx.png всегда разный, на каждом есть текст, указывающий путь в файловой системе – так мы будем знать, какое правило сработало и какой файл nginx.png нам отдает веб-сервер. Чтобы выполнять http-запросы, обращаясь к веб-серверу по имени домена example.com – добавим запись в файл hosts на том компе, с которого будем обращаться к веб-серверу.

Первый пример

Выполняем http-запрос http://example.com/img/logo/nginx.png

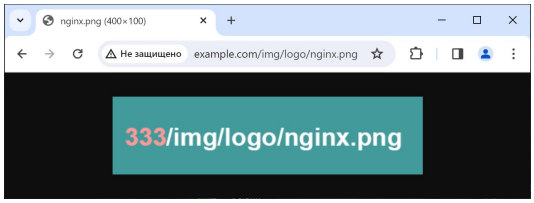
```
$ sudo nano /etc/nginx/sites-available/example.com
```

```
server {
    listen 80;
    server_name example.com;

    root /var/www/example.com;

    location /img {
        root /var/www/example.com/111;
    }
    location /img/logo {
        root /var/www/example.com/222;
    }
    location ~ ^/img {
        root /var/www/example.com/333;
    }
    location ~ ^/img/logo {
        root /var/www/example.com/444;
```

```
    }
}
```

```
$ sudo nano systemctl restart nginx.service
```
Копировать

Nginx сначала ищет самое длинное совпадение – это будет второй блок location. Поскольку самое длинное совпадение не имеет префикса ^~ – nginx переходит к поиску по регулярным выражениям. Причем, останавливается на первом совпадении – это третий блок. Чтобы сработал четвертый блок – его надо разместить перед третьим.



Второй пример

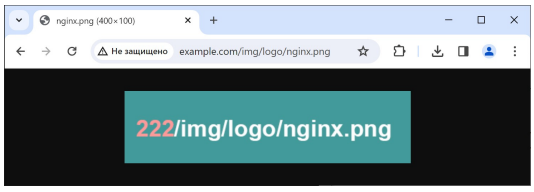
Выполняем http-запрос http://example.com/img/logo/nginx.png

```
$ sudo nano /etc/nginx/sites-available/example.com
```
Копировать

```
server {
    listen 80;
    server_name example.com;

    root /var/www/example.com;

    location /img {
        root /var/www/example.com/111;
    }
    location ^~ /img/logo {
        root /var/www/example.com/222;
    }
    location ~ ^/img {
        root /var/www/example.com/333;
    }
    location ~ ^/img/logo {
        root /var/www/example.com/444;
    }
}
```
Копировать

```
$ sudo nano systemctl restart nginx.service
```
Копировать

Nginx сначала ищет самое длинное совпадение – это будет второй блок location. Поскольку есть префикс ^~ – поиск завершается.



Третий пример

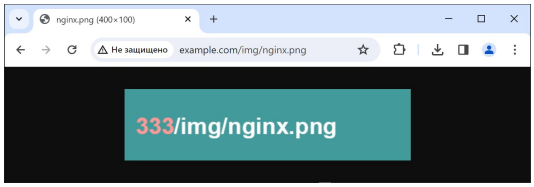
Выполняем http-запрос http://example.com/img/nginx.png

```
$ sudo nano /etc/nginx/sites-available/example.com
```
Копировать

```
server {
    listen 80;
    server_name example.com;

    root /var/www/example.com;

    location /img {
        root /var/www/example.com/111;
    }
    location /img/logo {
        root /var/www/example.com/222;
    }
    location ~ ^/i {
        root /var/www/example.com/333;
    }
    location ~ ^/img {
        root /var/www/example.com/444;
    }
}
```
Копировать

```
$ sudo nano systemctl restart nginx.service
```
Копировать

Nginx сначала ищет самое длинное совпадение – это будет первый блок location. Поскольку самое длинное совпадение не имеет префикса ^~ – nginx переходит к поиску по регулярным выражениям. Причем, останавливается на первом совпадении – это третий блок. Чтобы сработал четвертый блок – его надо разместить перед третьим.



Четвертый пример

Выполняем http-запрос http://example.com/logo/nginx.png

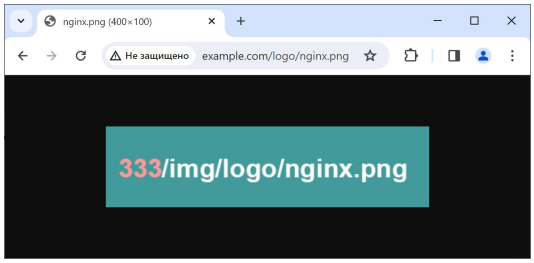
```
$ sudo nano /etc/nginx/sites-available/example.com
```
Копировать

```
server {
    listen 80;
    server_name example.com;

    root /var/www/example.com;

    location /logo {
        root /var/www/example.com/111/img;
    }
    location /img/logo {
        root /var/www/example.com/222;
    }
    location ~ ^/lo {
        root /var/www/example.com/333/img;
    }
    location ~ ^/logo {
```
Копировать

```
        root /var/www/example.com/444/img;
    }
}
```

```
$ sudo nano systemctl restart nginx.service
```
Копировать

Nginx сначала ищет самое длинное совпадение – это будет первый блок location. Поскольку самое длинное совпадение не имеет префикса ^~ – nginx переходит к поиску по регулярным выражениям. Причем, останавливается на первом совпадении – это третий блок. Чтобы сработал четвертый блок – его надо разместить перед третьим.



Примечание

Выбрал не слишком удачный способ посмотреть, какой блок location сработал – через текст на всех изображениях nginx.png. Все это можно сделать намного проще – если внутри каждого блока location разместить директиву add_header. Тогда через инструменты разработчика DevTools браузера можно будет посмотртеь заголовки и узнать, какой блок сработал.

Внутренние редиректы

Вернемся опять к файлу конфигурации nginx – рассмотрим директивы, которые могут вызывать внутренние редиректы. Это значит, что когда nginx нашел подходящий location, то директивы внутри этого location могут предписывать начать новые поиск подходящего location.

Директива index

Директива index всегда вызывает внутреннюю переадресацию, если используется для обработки запроса. Точные совпадения location = string {...} часто используются для ускорения поиска с немедленным завершением алгоритма. Однако, если точное совпадение расположения представляет собой каталог, есть вероятность, что запрос будет переадресован в другой location.

В примере ниже первому расположению соответствует URI запроса /exact, но унаследованная директива index активирует внутренний редирект во второй блок.

```
server {
    listen 80;
    server_name example.com www.example.com;

    root /var/www/example.com;
    index index.html;

    location = /exact {
```
Копировать

```
        # ...прочие директивы конфигурации...
    }
    location / {
        # ...прочие директивы конфигурации...
    }
}
```

Чтобы этого избежать – можно переопределить директиву index, установив для нее значение, которое никогда не совпадет. И можно включить директиву autoindex для показа содержимого директории вместо 404 Not Found.

Копировать

```
server {
    listen 80;
    server_name example.com www.example.com;

    root /var/www/example.com;
    index index.html;

    location = /exact {
        index nothing_will_match;
        autoindex on;
    }
    location / {
        # ...прочие директивы конфигурации...
    }
}
```

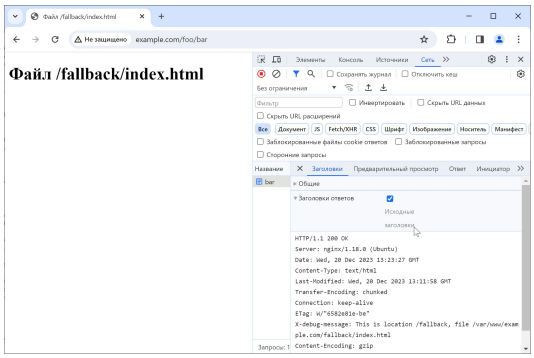
Директива try_files

Директива предписывает nginx проверить существование набора файлов или каталогов с определенным именем. Последним параметром может быть URI, на который nginx осуществляет внутреннюю переадресацию, если подходящий файл или директория не существует.

Копировать

```
server {
    listen 80;
    server_name example.com www.example.com;

    root /var/www/example.com;
    index index.html;

    location / {
        try_files $uri $uri/ /fallback/index.html;
    }

    location /fallback {
        add_header X-debug-message "This is location /fallback
        # ...прочие директивы конфигурации...
    }
}
```

При http-запросе URI /foo/bar – nginx найдет первый блок location и начнет выполнять директиву try_files. Сперва проверит существование файла /var/www/example.com/foo/bar, потом существование директории /var/www/example.com/foo/bar. Поскольку нет ни файла, ни директории – выполнит внутренний редирект. Найдет новый location и отправит клиенту файл /var/www/example.com/fallback/index.html.

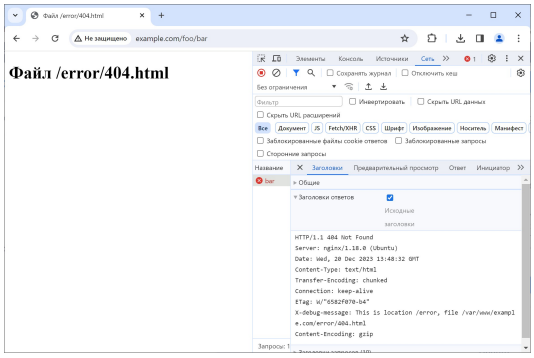


Директива error_page

Директива используется, чтобы определить, что должно происходить при получении определенных кодов состояния, например 404 или 500.

```
server {
    listen 80;
    server_name example.com www.example.com;

    root /var/www/example.com;
    index index.html;

    location / {
        error_page 404 /error/404.html;
    }

    location /error {
        add_header X-debug-message "This is location /error, f
        # ...прочие директивы конфигурации...
    }
}
```

Каждый запрос, кроме начинающихся с `/error`, будет обрабатываться первым блоком, который будет выводить файлы из `/var/www/example.com`. Однако, если файл не найден (статус 404) – будет выполнена внутренняя переадресация. Nginx найдет новый `location` и отправит клиенту файл `/var/www/example.com/error/404.html`.



Установка службы PHP-FPM

Нужно установить пакет `php8.1-fpm`. FPM расшифровывается как Fastcgi Process Manager, менеджер процессов FastCGI. PHP-FPM запускается как отдельный процесс и взаимодействует с веб-сервером через порт 9000 или сокетный файл. Является альтернативной реализацией PHP FastCGI с несколькими

дополнительными возможностями, обычно используемыми для высоконагруженных сайтов.

```
$ sudo apt update
$ sudo apt install php8.1-fpm
```

Проверяем работу службы PHP-FPM

```
$ systemctl is-active php8.1-fpm.service
active
```

Файл конфигурации службы – это /etc/php/8.1/fpm/php-fpm.conf, мы его трогать не будем – это тема для отдельного разговора

```
;;;;;;;;;;;;;;;;;;;;
; FPM Configuration ;
;;;;;;;;;;;;;;;;;;;;

; All relative paths in this configuration file are relative
; prefix (/usr). This prefix can be dynamically changed by u
; '-p' argument from the command line.

;;;;;;;;;;;;;;;;;;
; Global Options ;
;;;;;;;;;;;;;;;;;;

[global]
; Pid file
; Note: the default prefix is /var
; Default Value: none
; Warning: if you change the value here, you need to modify
; service PIDFile= setting to match the value here.
pid = /run/php/php8.1-fpm.pid

; Error log file
; If it's set to "syslog", log is sent to syslogd instead of
; into a local file.
```

Подключение Nginx к PHP-FPM

Чтобы принимать запросы FastCGI от Nginx, PHP-FPM может прослушивать сокет TCP/IP или UNIX сокет. Сокеты UNIX являются средством межпроцессного взаимодействия, которое обеспечивает эффективный обмен данными между процессами, работающими на одном сервере, в то время как сокеты TCP/IP позволяют процессам обмениваться данными по сети.

В отличие от сокета TCP/IP, который идентифицирует сервер по ip-адресу и порту (например, 127.0.0.1:9000), можно привязать сервер к сокету UNIX, используя путь к файлу (например, /run/php-fpm/www.sock), который виден в файловой системе.

Сокет UNIX – это особый тип файла – к нему применяются разрешения на доступ к файлам и каталогам (как в случае с любым другим типом файла), и его можно использовать для ограничения того, какие процессы на хосте могут читать и записывать в файл (и таким образом, общаться с внутренним сервером).

Таким образом, сокет UNIX является безопасным, поскольку его могут использовать только процессы на локальном хосте. Сокет TCP/IP может быть доступен из Интернета, и это может представлять угрозу безопасности, если не будут приняты дополнительные меры безопасности, такие как настройка брандмауэра.

Файл конфигурации nginx для работы с php-fpm

```
$ sudo nano /etc/nginx/sites-available/example.com
```

```
server {
    listen 80;
    server_name example.com www.example.com;

    root /var/www/example.com/;
    index index.php index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        # файл сокета для общения с php-fpm службой, которая б
        fastcgi_pass unix:/run/php/php8.1-fpm-example-com.sock
        # может быть файл /etc/nginx/fastcgi_params или файл /
        # у файлов одинаковое содержимое, но в fastcgi.conf ec
        include fastcgi.conf;
    }
}
```

```
$ sudo systemctl restart nginx.service
```

В файле /etc/nginx/fastcgi.conf устанавливаются значения переменных, которые будут доступны php скрипту во время выполнения.

```
$ cat /etc/nginx/fastcgi.conf
```

```
fastcgi_param  SCRIPT_FILENAME    $document_root$fastcgi_scr
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  SCRIPT_NAME        $fastcgi_script_name;
fastcgi_param  REQUEST_URI        $request_uri;
fastcgi_param  DOCUMENT_URI       $document_uri;
fastcgi_param  DOCUMENT_ROOT      $document_root;
fastcgi_param  SERVER_PROTOCOL    $server_protocol;
fastcgi_param  REQUEST_SCHEME     $scheme;
fastcgi_param  HTTPS              $https if_not_empty;

fastcgi_param  GATEWAY_INTERFACE  CGI/1.1;
fastcgi_param  SERVER_SOFTWARE    nginx/$nginx_version;

fastcgi_param  REMOTE_ADDR        $remote_addr;
fastcgi_param  REMOTE_PORT        $remote_port;
fastcgi_param  REMOTE_USER        $remote_user;
```

```
fastcgi_param  SERVER_ADDR        $server_addr;
fastcgi_param  SERVER_PORT        $server_port;
fastcgi_param  SERVER_NAME        $server_name;
```

Файл fastcgi_params появился намного раньше fastcgi.conf. Часто можно встетить конфигурации, где используется именно fastcgi_params, а значение переменной SCRIPT_FILENAME устанавливается отдельно внутри location для FastCGI.

Создадим еще файл /var/www/example.com/index.php для проверки работы php

Копировать
```
$ sudo nano /var/www/example.com/index.php
```

Копировать
```
<h1>Сайт example.com, файл /index.php</h1>
<?php phpinfo(); ?>
```

Настройка PHP-FPM для прослушивания сокета UNIX

После установки PHP-FPM есть файл конфигурации пула www.conf для обслуживания запросов от Nginx или Apache. В PHP-FPM каждый запрос обрабатывается заранее запущенным процессом из пула. Потому что запуск процесса – дорогая операция, лучше запустить все процессы заранее. Все процессы в пуле – однотипные, предназначены для обработки запросов от одного сайта. Для нашего сайта example.com мы создадим отдельный пул.

Копировать
```
$ sudo cp /etc/php/8.1/fpm/pool.d/www.conf /etc/php/8.1/fpm/po
```

И отредактируем файл конфигурации пула, изменим только имя и директиву listen. Настройка пула – это тема для отдельного разговора.

Копировать
```
$ sudo nano /etc/php/8.1/fpm/pool.d/example-com.conf
```

Копировать
```
; имя пула, должно быть обязательно задано и быть уникальным
[example-com]

; пользователь и группа, от имени которого работают процессы
user = www-data
group = www-data

listen = /run/php/php8.1-fpm-example-com.sock

; пользователь и группа, которые могут читать и записывать uni
; здесь указываются пользователь и группа, под которым работае
listen.owner = www-data
listen.group = www-data
listen.mode = 0660
```

Вообще, было бы правильно создать отдельного пользователя, от имени которого PHP-FPM запускал бы процессы для этого пула. Но упростим себе немного задачу – пусть процессы запускаются от имени уже существующего пользователя www-data.

Файл пула www.conf нужно удалить или переименовать – чтобы этот пул не создавался и мы не тратили ресурсы сервера впустую.

```
$ sudo mv /etc/php/8.1/fpm/pool.d/www.conf /etc/php/8.1/fpm/po
```

Перезапускаем службу, чтобы применить новые настройки

```
$ sudo systemctl restart php8.1-fpm.service
```

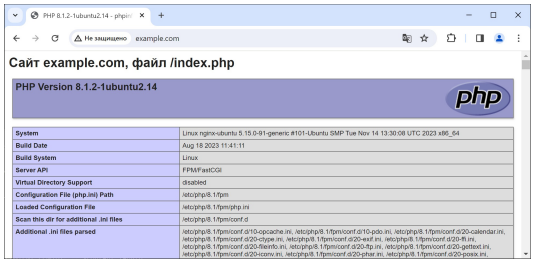
Проверяем работу службы PHP-FPM с новыми настройками

```
$ systemctl is-active php8.1-fpm.service
active
```

Смотрим запущенные процессы пользователя www-data

```
$ ps -u www-data -F
UID          PID    PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
www-data     673     669  0 13975  5708   0 12:09 ?        00:00:00 php-fpm: pool www
www-data    1387    1386  0 50793  7128   0 14:20 ?        00:00:00 php-fpm: pool example-com
www-data    1388    1386  0 50793  7128   0 14:20 ?        00:00:00 php-fpm: pool example-com
```

Их всего три, один процесс nginx и два процесса php-fpm. Все готово, можно проверять



Настройка PHP-FPM для прослушивания сокета TCP/IP

Отличий от прослушивания сокета UNIX немного – нужно изменить директиву listen файла конфигурации пула процессов php-fpm и директиву fastcgi_pass файла конфигурации nginx.

```
$ sudo nano /etc/php/8.1/fpm/pool.d/example-com.conf
```

```
; имя пула, должно быть обязательно задано и быть уникальным
[example-com]

; пользователь и группа, от имени которого работают процессы
user = www-data
group = www-data

listen = 127.0.0.1:9000

; пользователь и группа, которые могут читать и записывать uni
; здесь указываются пользователь и группа, под которым работае
listen.owner = www-data
```

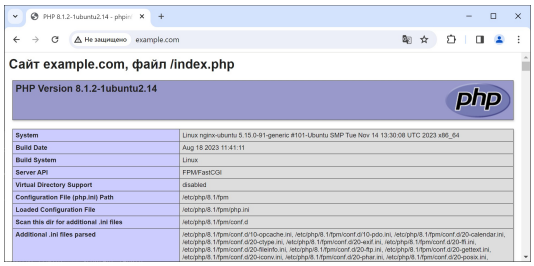
```
listen.group = www-data
listen.mode = 0660
```

Копировать
```
$ sudo systemctl restart php8.1-fpm.service
```

Копировать
```
$ sudo nano /etc/nginx/sites-available/example.com
```

Копировать
```
server {
    listen 80;
    server_name example.com www.example.com;
    root /var/www/example.com/;

    index index.php index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        # файл сокета для общения с php-fpm службой, которая б
        fastcgi_pass 127.0.0.1:9000;
        # может быть файл /etc/nginx/fastcgi_params или файл /
        # у файлов одинаковое содержимое, но в fastcgi.conf ec
        include fastcgi.conf;
    }
}
```

Копировать
```
$ sudo systemctl restart nginx.service
```



Файл конфигурации Nginx для PHP-FPM

Мы до сих пор использовали минимальную конфигурацию для работы nginx с php-fpm. Давайте доработаем конфигурацию, чтобы она больше отвечала реальности.

```
server {
    listen 80;
    server_name example.com www.example.com;

    root /var/www/example.com/;
    index index.php index.html;

    # при возникновении ошибок будем показывать специальные
    error_page 404 403 /error/404.html;
    error_page 403 /error/403.html;
    error_page 500 502 503 504 /error/50x.html;

    location / {
        try_files $uri $uri/ =404;
```

```
    }

    location ~ \.php$ {
        # файл сокета для общения с php-fpm службой, которая
        fastcgi_pass unix:/run/php/php8.1-fpm-example-com.so
        # может быть файл /etc/nginx/fastcgi_params или файл
        # у файлов одинаковое содержимое, но в fastcgi.conf
        include fastcgi.conf;
    }
```

Директива fastcgi_split_path_info определяет регулярное выражение с двумя захваченными группами. Первая захваченная группа используется в качестве значения переменной $fastcgi_script_name. Вторая захваченная группа используется качестве значения переменной $fastcgi_path_info. Необходимост в этой директиве возникает, если запрошенный URI имеет вид /index.php/foo/bar.

Директива fastcgi_index не нужна, если службы nginx.service и php8.1-fpm.service работают на одном сервере. В этом случае nginx при запросе директории может проверить существование индексного файла index.php в файловой системе сервера. Но есл служба php8.1-fpm.service работает на другом хосте – такой fastcgi_index – ее значение просто добавляется к любой запрошенной директории.

Копировать

```
server {
    location ~ (\.php|/)$ {
        fastcgi_pass 192.168.100.2:9000;
        fastcgi_index  index.php;
    }
}
```

Настройка поддержки HTTPS

У нас тестовый веб-сервер, нужно выпустить сертифкат для локального домена example.com. Это делается в несколько этапов. Сначала создаем приватный и публичный ключ Центра Сертификации (CA, Certificate Authority). Потом публичный ключ CA подписываем приватным ключом CA. Таким образом получаем самоподписанный сертификат CA, который надо добавить в доверенные в браузере. Далее создаем приватный и публичный ключ для домена example.com. Публичный ключ домена example.com подписываем приватным ключом CA – получаем сертификат для домена. *Чтобы подписать публичный ключ – нужно создать запрос на подпись.*

Копировать

```
$ mkdir ~/ca
$ chdir ~/ca
```

Для создания пары ключей и подписи будем использовать утилиту openssl

Копировать

```
$ openssl команда опции
```

Создание приватного ключа Центра Сертификации

```
$ openssl genpkey \
>    -aes256 \
>    -algorithm RSA \
>    -pkeyopt rsa_keygen_bits:4096 \
>    -pass pass:qwerty \
>    -out root-ca.key
```

Команда genpkey (заменяет genrsa, gendh и gendsa) означает создание приватного ключа (generate private key). Алгоритм ключа RSA, дина 4096 бит, зашифрован алгоритмом aes256. Опция -pass задает пароль qwerty для обеспечения безопасности ключа. Опция -out указывает на имя файла для сохранения, без этой опции файл будет выведен в стандартный вывод.

Создание самоподписанного корневого сертификата

```
$ openssl req \
>    -x509 \
>    -new \
>    -key root-ca.key \
>    -pass pass:qwerty \
>    -days 7300 \
>    -subj "/C=RU/ST=Moscow/L=Moscow/O=Demo Inc/OU=IT Dept/CN=
>    -addext "basicConstraints = critical,CA:TRUE" \
>    -addext "subjectKeyIdentifier = hash" \
>    -addext "authorityKeyIdentifier = keyid:always,issuer" \
>    -addext "keyUsage = critical,keyCertSign,cRLSign" \
>    -out root-ca.crt
Enter pass phrase for root-ca.key: qwerty
```

Просмотр самоподписанного сертификата (не обязательно, просто для проверки, что все правильно)

```
$ openssl x509 -text -noout -in root-ca.crt
```

Команда req означает запрос на подпись (CSR, Certificate Signing Request). Опция -new означает создание нового сертификата. Опция -key указывает на имя файла приватного ключа. Опция -days – срок действия сертификата (три года).

Создание файла приватного ключа и файла запроса на подпись (сразу две операции одной командой)

```
$ openssl req -newkey rsa:3072 \
>    -keyform PEM \
>    -outform PEM \
>    -nodes \
>    -subj "/C=RU/ST=Moscow/L=Moscow/O=Demo Inc/OU=IT Dept/CN=
>    -addext "basicConstraints = CA:FALSE" \
>    -addext "subjectKeyIdentifier = hash" \
>    -addext "keyUsage=digitalSignature,nonRepudiation,keyEnci
>    -addext "extendedKeyUsage=serverAuth" \
>    -addext "subjectAltName=DNS:example.com,DNS:www.example.c
>    -keyout example-com.key \
>    -out example-com.csr
```

Просмотр запроса на подпись (не обязательно, просто для проверки, что все правильно)

```
$ openssl req -text -noout -in example-com.csr
```

Подпись запроса и получение сертификата на год

```
$ openssl x509 -req \
>    -in example-com.csr \
>    -CA root-ca.crt \
>    -CAkey root-ca.key \
>    -CAcreateserial \
>    -days 365 \
>    -out example-com.crt \
>    --copy_extensions=copyall
Certificate request self-signature ok
subject=C = RU, ST = Moscow, L = Moscow, O = Demo Inc, OU = IT
Enter pass phrase for root-ca.key: qwerty
```

Просмотр подписанного сертификата (не обязательно, просто для проверки, что все правильно)

```
$ openssl x509 -text -noout -in example-com.crt
```

Скопируем приватный ключ и сертификат в директорию /etc/ssl

```
$ sudo cp /home/evgeniy/ca/example-com.crt /etc/ssl/certs/
$ sudo cp /home/evgeniy/ca/example-com.key /etc/ssl/private/
```

Отредактируем файл конфигурации nginx

```
$ nano /etc/nginx/sites-available/example.com
```

```
server {
    listen 80;
    listen 443 ssl;
    server_name example.com www.example.com;

    ssl_certificate /etc/ssl/certs/example-com.crt;
    ssl_certificate_key /etc/ssl/private/example-com.key;

    root /var/www/example.com/;
    index index.php index.html;

    error_page 404 403 /error/404.html;
    error_page 500 502 503 504 /error/50x.html;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        # файл сокета для общения с php-fpm службой, которая
        fastcgi_pass 127.0.0.1:9000;
        # может быть файл /etc/nginx/fastcgi_params или файл
        # у файлов одинаковое содержимое, но в fastcgi_param
```

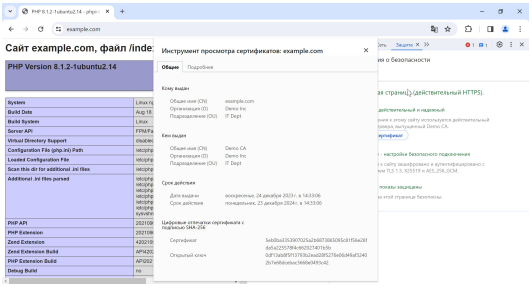
```
$ sudo systemctl restart nginx.service
```

Теперь нужно скопировать корневой сертификат /home/evgeniy/ca/root-ca.crt на свой компьютер и добавить его в доверенные в браузере.

```
$ scp evgeniy@example.com:/home/evgeniy/ca/root-ca.crt ~/
```

Копировать

В браузере Chrome нужно зайти в настройки, перейти в «Безопасность и конфиденциальность», потом «Безопасность», дальше «Настроить сертификаты». И добавить сертификат в «Доверенные корневые центры сертификации».



Редиректы с кодом 301

Для сайта нужно настроить 301 редирект с домена www.example.com на домен example.com. Или наоборот, с домена example.com на домен www.example.com. Кроме того, нужно настроить редирект с http на https. И желательно – редирект с /some/path/index.php на /some/path/.

Здесь возможны различные конфигурации, вот два варианта

Копировать

```
server {
    listen 80;
    server_name example.com www.example.com;
    # редирект с http на https
    return 301 https://example.com$request_uri;
}

server {
    listen 443;
    server_name example.com www.example.com;

    # редирект с www.example.com на example.com
    if ($host = www.example.com) {
        return 301 https://example.com$request_uri;
    }

    # редирект с /some/path/index.php на /some/path/
    if ($request_uri ~ "^(.*)index\.(php|html)") {
        return 301 $1;
    }

    ssl_certificate /etc/ssl/certs/example-com.crt;
    ssl_certificate_key /etc/ssl/private/example-com.key;
```

```
server {
    listen 80;
    server_name example.com www.example.com;
    # редирект с http на https
    return 301 https://example.com$request_uri;
```

```
}

server {
    listen 443 ssl;
    server_name www.example.com;

    ssl_certificate /etc/ssl/certs/example-com.crt;
    ssl_certificate_key /etc/ssl/private/example-com.key;

    # редирект с www.example.com на example.com
    return 301 https://example.com$request_uri;
}

server {
    listen 443 ssl;
    server_name example.com;

    # редирект с /some/path/index.php на /some/path/
```

Дополнительно

- Как установить Nginx на Ubuntu: пошаговая инструкция
- Оптимизация NGINX, ускорение работы сайта
- Директива location в Nginx, алгоритм выбора блока
- Размещение нескольких сайтов на NGINX и PHP-FPM в Ubuntu 14.04
- OpenSSL, часть 1: Создание собственного Удостоверяющего Центра
- OpenSSL, часть 2: Создание ключа и сертификата для веб-сервера
- OpenSSL, часть 3: Создание ключа и сертификата для клиента

Похожие записи

- Монтирование NFS на сервере Ubuntu 18.04 LTS
- Установка vnc4server на Ubuntu Server 18.04 LTS
- Установка DHCP-сервера на Ubuntu Server 18.04 LTS
- Локализация Ubuntu Server 18.04 LTS
- Установка SSH-сервера на Ubuntu 18.04 LTS
- Ubuntu. Установка и настройка supervisor
- Установка WireGuard на Ubuntu 20.04 LTS. Часть вторая из двух

Поиск: CLI · Nginx · Конфигурация · Настройка · Сервер · Установка

Узелки на память: Web-разработка, 1С:Предприятие, ОС Linux