????????? ? Linux

Установка и настройка различных программ и сервисов Linux

Nginx

Настройки и примеры конфигурации Nginx

?????????? ? ????????????

??? 1: ????????? Nginx

Обновите список пакетов и установите Nginx из стандартных репозиториев Debian/Ubuntu.

```
sudo apt update && sudo apt install -y nginx
```

??? 2: ?????????? ??????? Nginx

Основные команды для управления состоянием службы Nginx.

Проверить статус:

```
sudo systemctl status nginx
```

Добавить в автозагрузку:

```
sudo systemctl enable nginx
```

Перезапустить Nginx (требуется после изменения конфигураций):

```
sudo systemctl restart nginx
```

??? 3: ????????? ??????????? (iptables)

Чтобы Nginx был доступен извне, необходимо открыть порты 80 (HTTP) и 443 (HTTPS). Для сохранения правил после перезагрузки рекомендуется использовать пакет `iptables-persistent`.

Сначала установите пакет для сохранения правил:

```
sudo apt install -y iptables-persistent
```

Затем добавьте правила для портов 80 и 443. `-I INPUT 5` вставляет правило на 5-ю позицию. Номер может потребоваться изменить в зависимости от вашей конфигурации.

```
sudo iptables -I INPUT 5 -p tcp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
```

```
sudo iptables -I INPUT 6 -p tcp --dport 443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT
```

Наконец, сохраните текущие правила, чтобы они применялись после перезагрузки:

```
sudo netfilter-persistent save
```

?????? ???????? ????? /etc/nginx/nginx.conf

Это основной конфигурационный файл, который задает глобальные параметры работы веб-сервера. Редактировать его нужно с осторожностью.

```
# Указывает пользователя, от имени которого будут работать рабочие процессы Nginx.
# Рекомендуется использовать специального пользователя (www-data) для безопасности.
user www-data;
# Количество рабочих процессов. 'auto' — оптимальный вариант, Nginx сам определит
# количество, равное числу ядер процессора.
worker_processes auto;
# Путь к файлу, где будет храниться PID главного процесса Nginx.
pid /run/nginx.pid;
# Включает файлы с дополнительными модулями, если они установлены.
include /etc/nginx/modules-enabled/*.conf;
events {
    # Максимальное количество одновременных соединений, которое может
    # обработать один рабочий процесс. Общий лимит = worker_processes * worker_connections.
    worker_connections 768;
    # multi_accept on; # Позволяет рабочему процессу принимать все новые соединения сразу.
}
http {
    # Базовые настройки для обработки MIME-типов файлов.
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    # server_tokens off; # Скрывает версию Nginx в ответах сервера.
    # Путь к файлу с MIME-типами.
    include /etc/nginx/mime.types;
    # MIME-тип по умолчанию.
    default_type application/octet-stream;
    # Настройки для логирования.
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    # Настройки для Gzip-сжатия. Уменьшает размер передаваемых данных.
```

```
    gzip on;
    # gzip_vary on;
    # gzip_proxied any;
    # gzip_comp_level 6;
    # gzip_buffers 16 8k;
    # gzip_http_version 1.1;
    # gzip_types text/plain text/css application/json ...;
    # Подключение конфигурационных файлов для каждого отдельного сайта (виртуального хоста).
    # Это главная директива, которая делает конфигурацию модульной.
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

SSL/TLS ? Let's Encrypt

??? 1: ????????? Certbot

Certbot — это инструмент для автоматического получения и обновления сертификатов Let's Encrypt. Установите его и плагин для Nginx.

```
sudo apt install -y certbot python3-certbot-nginx
```

??? 2: ????????? ???????????

Запустите Certbot для вашего домена. Он автоматически изменит вашу конфигурацию Nginx, чтобы использовать HTTPS.

```
sudo certbot --nginx -d example.com -d www.example.com
```

??? 3: ???????? ?????????????

Certbot автоматически настраивает системный таймер (в systemd или cron), который будет продлевать сертификаты за 30 дней до их истечения. Вы можете протестировать этот процесс.

```
sudo certbot renew --dry-run
```

??????? ???????????? HTTPS

1. Стандартная конфигурация от Certbot

Это базовый и надежный вариант, который Certbot создает автоматически. Он указывает пути к сертификатам и подключает рекомендованные Certbot'ом параметры безопасности.

```
server {
    listen 443 ssl http2;
    server_name example.com www.example.com;
    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
        # ... остальная конфигурация (root, location)
```

```
}
```

2. Редирект с HTTP на HTTPS

Это обязательный блок для любого сайта на HTTPS. Он "слушает" небезопасный 80-й порт и, получив запрос, немедленно отправляет клиенту ответ с кодом 301 (Permanent Redirect), указывая новый адрес на `https://`. Переменные `$host` и `$request_uri` сохраняют исходный домен и путь, чтобы пользователь попал на нужную страницу, но уже по защищенному протоколу.

```
server {
    listen 80;
    server_name example.com www.example.com;
    return 301 https://$host$request_uri;
}
```

3. Повышенная безопасность (Strong Ciphers + TLS 1.3)

Здесь мы явно указываем, какие протоколы и шифры разрешено использовать. `TLSv1.2` и `TLSv1.3` — современные безопасные протоколы. Длинный список `ssl_ciphers` перечисляет надежные наборы шифров и задает их приоритет. Это защищает от использования устаревших и уязвимых алгоритмов шифрования.

```
server {
    # ...
    listen 443 ssl http2;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384;
    # ...
}
```

4. С HSTS (HTTP Strict Transport Security)

Это мощный механизм защиты. Когда пользователь впервые заходит на сайт, сервер отправляет заголовок `Strict-Transport-Security`. Браузер пользователя "запоминает" эту инструкцию и в течение указанного времени (`max-age` в секундах, здесь — 2 года) будет автоматически и принудительно выполнять все запросы к этому домену (и поддоменам, если указано `includeSubDomains`) только по HTTPS, даже если пользователь введет в адресной строке `http://`.

```
server {
    # ...
    listen 443 ssl http2;
    # ...
    add_header Strict-Transport-Security "max-age=63072000; includeSubDomains; preload"
always;
    # ...
}
```

5. Оптимизация производительности SSL

Установка SSL-соединения — ресурсоемкий процесс. Чтобы ускорить повторные подключения от одного и того же клиента, Nginx может кэшировать параметры сессии. `ssl_session_cache` создает в памяти кэш размером 10 МБ, а `ssl_session_timeout` указывает, как долго хранить сессию. Это значительно снижает нагрузку на сервер и ускоряет загрузку сайта для вернувшихся посетителей.

```
server {
    # ...
    listen 443 ssl http2;
    # ...
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 1d;
    ssl_session_tickets off; # Рекомендуется для Perfect Forward Secrecy
    # ...
}
```

6. Полная конфигурация с лучшими практиками

Этот пример объединяет все лучшее: указывает пути к сертификатам, использует современные протоколы и шифры, добавляет заголовок HSTS для защиты и включает кэширование сессий для производительности. Это хороший, сбалансированный шаблон для большинства современных сайтов.

```
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name example.com;
    # SSL
    ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
    ssl_trusted_certificate /etc/letsencrypt/live/example.com/chain.pem;
```

```
    # Security
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384';
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
    # Performance
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 1d;
    root /var/www/example.com;
    index index.html;
    # ...
}
```

Nginx

????????????? ??? ??????

????????? ?????????: ?? ???????? ?? ????????? ?? ?????? ?????

После установки Nginx по умолчанию активен один сайт, который вы видите на 80 порту — это страница "Welcome to nginx!". Его конфигурация находится в файле `/etc/nginx/sites-enabled/default`. Чтобы разместить свой сайт, нужно отключить эту конфигурацию и создать свою.

??? 1: ?????????? ????? ?? ?????????

Удалим символическую ссылку на конфигурацию по умолчанию. Это безопаснее, чем удалять сам файл, так как он останется в `/etc/nginx/sites-available/` в качестве примера.

```
sudo rm /etc/nginx/sites-enabled/default
```

??? 2: ???????? ???????????? ??? ?????? ?????

Создадим новый файл конфигурации для нашего домена (например, `example.com`) в каталоге `sites-available`.

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

Вставьте в этот файл базовую конфигурацию для статического сайта. Не забудьте создать каталог `/var/www/example.com` и положить туда файлы вашего сайта (например, `index.html`).

```
server {
    listen 80;
    server_name example.com www.example.com;
    root /var/www/example.com;
    index index.html index.htm;
    location / {
        try_files $uri $uri/ =404;
    }
}
```

??? 3: ????????? ?????? ?????

Теперь создадим символическую ссылку из `sites-available` в `sites-enabled`, чтобы Nginx "увидел" и загрузил нашу конфигурацию.

```
sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/
```

??? 4: ???????? ? ?????????? ???????????

Перед перезапуском всегда проверяйте конфигурацию на наличие синтаксических ошибок.

Проверить конфигурацию:

```
sudo nginx -t
```

Если ошибок нет (вывод: `syntax is ok, test is successful`), применяем изменения командой `reload`, которая делает это без разрыва текущих соединений.

```
sudo systemctl reload nginx
```

??????? ???????????? ????????????

1. Reverse Proxy ? ???????????? ??????? ?? IP (??? ?????-??????)

Очень частый сценарий: защитить внутренний инструмент (например, Grafana, Prometheus, веб-интерфейс 1С), работающий на локальном порту (например, 3000), и разрешить доступ к нему только из офисной сети.

```
server {
    listen 80;
    server_name dashboard.example.com;
    # Блок ограничения доступа
    allow 88.77.66.55; # Ваш офисный/домашний IP
    allow 127.0.0.1;   # Доступ с самого сервера
    deny all;          # Запретить доступ всем остальным
    location / {
        proxy_pass http://localhost:3000; # Адрес вашего внутреннего приложения
        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_set_header X-Forwarded-Proto $scheme;
```

```
    }
}
```

2. ????????? ????????? ?????????

Позволяет просматривать содержимое каталога, если в нем отсутствует индексный файл. Полезно для организации файловых архивов.

```
location /downloads/ {
    root /var/www/files;
    autoindex on; # Включить листинг
    autoindex_exact_size off; # Показывать размеры файлов в удобном виде (KB, MB)
    autoindex_localtime on; # Показывать локальное время файлов
}
```

3. ?????? ??????? ? ??????? ??????

Блокирует доступ к файлам, начинающимся с точки, таким как `.htaccess` или `.git`.

```
location ~ /\. {
    deny all;
    return 404;
}
```

???????????? `.htaccess` ? Nginx

Важное отличие Nginx от Apache: **Nginx не поддерживает файлы `.htaccess`**. Это сделано осознанно для повышения производительности. Nginx считывает конфигурацию один раз при старте или перезагрузке и держит ее в памяти, в то время как Apache проверяет наличие `.htaccess` в каждом каталоге при каждом запросе.

Все правила, которые вы бы поместили в `.htaccess`, в Nginx прописываются непосредственно в конфигурационном файле вашего сайта (в блоке `server` или `location`). Вот как эмулировать популярные правила:

????????????? (Rewrite) ??? "????????" URL

Типичная задача для фреймворков: все запросы, которые не ведут к существующим файлам, отправлять на главный `index.php`.

```
# Пример для Apache (.htaccess)
# RewriteEngine On
# RewriteCond %{REQUEST_FILENAME} !-f
```

```
# RewriteCond %{REQUEST_FILENAME} !-d
# RewriteRule . /index.php [L]
# Эквивалент для Nginx
location / {
    try_files $uri $uri/ /index.php?$query_string;
}
```

?????? ??????? ? ????????

```
# Пример для Apache (.htaccess в папке /secret)
# deny from all
# Эквивалент для Nginx
location /secret/ {
    deny all;
    return 404; # Опционально, можно вернуть 403 или 404
}
```