Расширенный мини-курс по Nginx для начинающих

Полное руководство по синтаксису и конфигурациям Nginx

Создано для глубокого изучения веб-серверов и их настройки

Содержание

1 Что такое Nginx? 2
1.1 Зачем нужен Nginx? . . . 2

2 Установка Nginx 2
2.1 Шаги установки . . . 2

3 Базовая терминология 3

4 Синтаксис Nginx 3
4.1 Структура конфигурации . . . 3
4.2 Правила написания . . . 4
4.3 Типичные ошибки . . . 4

5 Базовая конфигурация 4
5.1 Пример конфигурации . . . 4
5.2 Подробный анализ . . . 5

6 Блок events 6
6.1 Пример конфигурации . . . 7
6.2 Подробный анализ . . . 7

7 Балансировка нагрузки 8
7.1 Пример конфигурации . . . 8
7.2 Подробный анализ . . . 8

8 Ограничение ресурсов 10
8.1 Ограничение скорости запросов . . . 10
8.2 Подробный анализ . . . 10

9 SSL/TLS 11
9.1 Пример конфигурации . . . 11
9.2 Подробный анализ . . . 11

10 Логирование 12
10.1 Пример конфигурации . . . 12
10.2 Подробный анализ . . . 13

11 Интеграция с Docker 13
11.1 Пример конфигурации . . . 13
11.2 Подробный анализ . . . 14

12 Команды Nginx 14

13 Полезные советы 14

1 Что такое Nginx?

Nginx — это высокопроизводительный веб-сервер, обратный прокси и кэширующий сервер с открытым исходным кодом. Он обслуживает веб-страницы, балансирует нагрузку, защищает приложения и ускоряет доставку контента. Представьте Nginx как диспетчера в аэропорту: он направляет запросы к нужным серверам, управляет потоками и предотвращает заторы.

1.1 Зачем нужен Nginx?

- **Производительность**: Обрабатывает тысячи запросов с минимальными ресурсами.
- **Универсальность**: Поддерживает статические сайты, динамические приложения, прокси и балансировку.
- **Гибкость**: Конфигурация адаптируется под любые задачи.
- **Надёжность**: Устойчив к высоким нагрузкам и атакам.

2 Установка Nginx

Установка Nginx проста. Пример для Ubuntu/Debian.

2.1 Шаги установки

1. Обновите пакеты:

```
    sudo apt update

```

2. Установите Nginx:

```
    sudo apt install nginx

```

3. Запустите Nginx:

```
    sudo systemctl start nginx

```

4. Включите автозапуск:

```
    sudo systemctl enable nginx

```

5. Проверьте статус:

```
    sudo systemctl status nginx

```

Что происходит? Nginx запускается на порту 80. Откройте `http://localhost` в браузере, чтобы увидеть приветственную страницу.

3 Базовая терминология

Для работы с Nginx важно знать:

Конфигурационный файл Файл (`/etc/nginx/nginx.conf`) с настройками сервера. Это как инструкция для Nginx.

Директива Команда, например, `listen` или `root`.

Контекст Блок (`http { }`, `server { }`) для группировки директив.

Виртуальный хост Настройка для разных доменов. Это как разные кассы в магазине.

Обратный прокси Перенаправление запросов. Это как секретарь, передающий звонки.

Upstream Группа серверов для балансировки. Это как команда работников.

Зона лимитов Память для лимитов (`limit_req_zone`).

4 Синтаксис Nginx

Конфигурации Nginx состоят из директив и контекстов, организованных иерархически. Понимание синтаксиса критично для написания корректных настроек.

4.1 Структура конфигурации

- **Директивы**: Простые команды вида `имя значение;`. Например:

```
    listen 80;

```

  Каждая директива заканчивается точкой с запятой (`;`). Пропуск `;` вызывает ошибку синтаксиса.

- **Контексты**: Блоки, содержащие директивы, обозначаются фигурными скобками `{ }`. Пример:

```
    server {
      listen 80;
    }

```

  Контексты могут быть вложенными: `http { server { location { } } }`.

- **Наследование**: Директивы из внешнего контекста (например, `http`) применяются к внутренним (`server`), если не переопределены. Например:

```
    http {
      access_log /var/log/nginx/access.log;
      server {
        # Лог унаследован
      }
    }

```

- **Комментарии**: Начинаются с # и игнорируются Nginx.

```
    # Это комментарий
    listen 80; # Слушать порт 80

```

4.2 Правила написания

- **Чувствительность к регистру**: Директивы чувствительны к регистру (`listen` ≠ `Listen`).
- **Пробелы**: Пробелы между именем директивы и значением обязательны:

```
    server_name example.com; # Корректно
    server_nameexample.com; # Ошибка

```

- **Экранирование**: Строки с пробелами или специальными символами заключаются в кавычки:

```
    auth_basic "Restricted Area";

```

- **Проверка**: Всегда проверяйте синтаксис:

```
    sudo nginx -t

```

4.3 Типичные ошибки

- Пропуск `;`: Приводит к ошибке `unexpected end of file`.
- Дублирование директив: Например, два `listen 80` в одном `server` вызывают конфликт.

- Неправильная вложенность: Директива вне нужного контекста (например, $proxy_{p}ass$`location`
Аналогия: Пишите конфигурацию как рецепт: каждый ингредиент (директива) на своём месте, шаги (контексты) чётко структурированы, иначе блюдо не получится.

5 Базовая конфигурация

Рассмотрим простой веб-сервер для статического сайта.

5.1 Пример конфигурации

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

5.2 Подробный анализ

- `http { }`:
  - **Что делает?** Определяет контекст для всех HTTP-серверов.
  - **Зачем?** Группирует настройки, связанные с HTTP-протоколом.
  - **Как работает?** Nginx обрабатывает все HTTP-запросы в этом контексте, передавая их вложенным блокам `server`.
  - **Ошибки**: Пропуск `http` для HTTP-настроек вызовет ошибку.
- `server { }`:
  - **Что делает?** Создаёт виртуальный хост для обработки запросов.
  - **Зачем?** Позволяет разделять настройки для разных доменов или портов.
  - **Как работает?** Nginx сопоставляет запросы с блоком `server` по `server_name` и `listen`.
- `listen 80;`:
  - **Что делает?** Указывает порт (80) для прослушивания HTTP-запросов.
  - **Зачем?** Определяет, на каком порту сервер принимает запросы.
  - **Как работает?** Nginx привязывается к порту 80, ожидая входящие соединения.
  - **Ошибки**: Если порт занят (например, другим сервером), Nginx не запустится.
- `server_name example.com www.example.com;`:
  - **Что делает?** Задаёт домены, которые обслуживает сервер.
  - **Зачем?** Позволяет отличать запросы к разным сайтам на одном сервере.
  - **Как работает?** Nginx сравнивает заголовок `Host` запроса с `server_name`, выбирая подходящий `server`.
  - **Ошибки**: Неправильный домен (например, опечатка) направит запросы к другому `server` или в никуда.
- `root /var/www/example.com/html;`:
  - **Что делает?** Указывает корневую папку для файлов сайта.
  - **Зачем?** Определяет, где искать файлы для ответа на запросы.
  - **Как работает?** Nginx добавляет путь запроса (например, `/index.html`) к `root`, формируя полный путь (`/var/www/example.com/html/index.html`).
  - **Ошибки**: Неправильный путь или отсутствие прав доступа вызовут ошибку 403/404.
- `index index.html;`:
  - **Что делает?** Задаёт файл по умолчанию для запросов к папкам.
  - **Зачем?** Упрощает доступ к страницам (например, `example.com/` возвращает `index.html`).
  - **Как работает?** Если запрос указывает папку, Nginx ищет `index.html` в этой папке.

  - **Ошибки**: Отсутствие файла `index.html` приведёт к 404, если не настроен fallback.

- `location / { }`:
  - **Что делает?** Определяет обработку запросов, начинающихся с `/` (все запросы).
  - **Зачем?** Позволяет настроить поведение для определённых путей.
  - **Как работает?** Nginx сопоставляет URI запроса с `location`, применяя вложенные директивы.

- `try_files $uri $uri/ /index.html;`:
  - **Что делает?** Проверяет существование файла или папки, возвращая `index.html` как fallback.
  - **Зачем?** Поддерживает одностраничные приложения (SPA), перенаправляя несуществующие пути к `index.html`.
  - **Как работает?** Nginx проверяет: - `$uri`: Файл (например, `/about` → `/var/www/example.com/html/about`). - `$uri/`: Папка с `index.html`. - `/index.html`: Фallback, если ничего не найдено.
  - **Ошибки**: Неправильный fallback (например, несуществующий файл) вызовет 404.

Аналогия: Эта конфигурация — как меню ресторана: `server` — стол, `listen` — номер стола, `server_name` — имя клиента, `root` — кухня, `index` — блюдо по умолчанию, `try_files` — запасной вариант, если блюдо недоступно.

Шаги:

1. Создайте папку: `sudo mkdir -p /var/www/example.com/html`.
2. Добавьте файл:

```
    echo "<h1Привет>, Nginx!</h1>" > /var/www/example.com/html/index.html

```

3. Сохраните конфигурацию в `/etc/nginx/sites-available/example.com`.
4. Активируйте:

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

```

5. Проверьте:

```
    sudo nginx -t

```

6. Перезагрузите:

```
    sudo systemctl reload nginx

```

6 Блок events

Блок `events { }` управляет обработкой соединений.

6.1 Пример конфигурации

```
events {
  worker_connections 2048;
  multi_accept on;
  use epoll;
}
```

6.2 Подробный анализ

- `events { }`:
  - **Что делает?** Определяет глобальные настройки обработки сетевых соединений.
  - **Зачем?** Контролирует, как Nginx взаимодействует с клиентами.
  - **Как работает?** Задаёт параметры для всех воркеров (процессов Nginx), управляющих соединениями.
  - **Ошибки**: Неправильные значения (например, слишком большое `worker_connection` могут исчерпать системные ресурсы.
- `worker_connections 2048;`:
  - **Что делает?** Устанавливает максимум одновременных соединений на воркер.
  - **Зачем?** Определяет, сколько клиентов может обслужить один процесс.
  - **Как работает?** Каждый воркер поддерживает до 2048 открытых соединений (включая запросы и ответы). Общее число соединений = `worker_processes` × `worker_connections`.
  - **Ошибки**: Слишком большое значение требует увеличения `ulimit -n` (лимит файловых дескрипторов).
- `multi_accept on;`:
  - **Что делает?** Разрешает воркеру принимать несколько соединений за раз.
  - **Зачем?** Ускоряет обработку входящих запросов.
  - **Как работает?** Вместо обработки одного соединения за цикл, воркер принимает все доступные соединения из очереди.
  - **Ошибки**: Может увеличить нагрузку на CPU при высоком трафике.
- `use epoll;`:
  - **Что делает?** Задаёт метод обработки соединений (`epoll` для Linux).
  - **Зачем?** Оптимизирует производительность на высоконагруженных серверах.
  - **Как работает?** `epoll` использует событийную модель, позволяя эффективно обрабатывать тысячи соединений с низкими затратами.
  - **Ошибки**: Использование `epoll` на не-Linux системах вызовет ошибку.

Аналогия: Блок `events` — как настройка конвейера на заводе: `worker_connections` — число деталей, которые может обработать рабочий, `multi_accept` — возможность брать несколько деталей сразу, `use epoll` — выбор эффективного станка.

Шаги:

1. Отредактируйте `/etc/nginx/nginx.conf`.
2. Проверьте:

```
    sudo nginx -t

```

3. Перезагрузите:

```
    sudo systemctl reload nginx

```

7 Балансировка нагрузки

Nginx распределяет запросы между серверами.

7.1 Пример конфигурации

```
http {
  upstream backend {
    ip_hash;
    server 192.168.1.101:3000 weight=2;
    server 192.168.1.102:3000;
    server 192.168.1.103:3000 max_fails=3 fail_timeout=30s;
  }
  server {
    listen 80;
    server_name app.example.com;
    location / {
      proxy_pass http://backend;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
    }
  }
}
```

7.2 Подробный анализ

- `upstream backend { }`:
  - **Что делает?** Определяет группу серверов для балансировки.
  - **Зачем?** Позволяет распределять нагрузку между несколькими бэкендами.
  - **Как работает?** Nginx направляет запросы к серверам в `upstream` по заданному алгоритму.
  - **Ошибки**: Неправильные адреса серверов или порты приведут к ошибкам проксирования.
- `ip_hash;`:
  - **Что делает?** Привязывает запросы от одного IP к одному серверу.
  - **Зачем?** Обеспечивает сохранение сессий (например, для авторизации).
  - **Как работает?** Nginx хеширует IP клиента и выбирает сервер на основе хеша.

  - **Ошибки**: Неэффективно, если IP клиентов меняются (например, мобильные сети).

- `server 192.168.1.101:3000 weight=2;`:
  - **Что делает?** Указывает сервер и его приоритет (`weight`).
  - **Зачем?** Позволяет направлять больше запросов к мощным серверам.
  - **Как работает?** Сервер получает вдвое больше запросов, чем другие (с `weight=1`).
  - **Ошибки**: Неправильный IP или порт сделают сервер недоступным.
- `server 192.168.1.103:3000 max_fails=3 fail_timeout=30s;`:
  - **Что делает?** Задаёт параметры отказоустойчивости.
  - **Зачем?** Исключает неработающие серверы из балансировки.
  - **Как работает?** После 3 неудачных попыток связи сервер исключается на 30 секунд.
- `proxy_pass http://backend;`:
  - **Что делает?** Перенаправляет запросы к группе `backend`.
  - **Зачем?** Связывает фронтенд (Nginx) с бэкендом (приложениями).
  - **Как работает?** Nginx отправляет запрос к одному из серверов `backend`, выбранному по `ip_hash`.
  - **Ошибки**: Если `backend` не определён, Nginx вернёт 502.
- `proxy_set_header Host $host;`:
  - **Что делает?** Передаёт заголовок `Host` бэкенду.
  - **Зачем?** Указывает бэкенду, к какому домену был запрос.
  - **Как работает?** Nginx копирует `Host` из запроса клиента (`app.example.com`).
  - **Ошибки**: Без этого бэкенд может неправильно обработать запрос.
- `proxy_set_header X-Real-IP $remote_addr;`:
  - **Что делает?** Передаёт IP клиента бэкенду.
  - **Зачем?** Позволяет бэкенду видеть настоящий IP, а не IP Nginx.
  - **Как работает?** Nginx добавляет заголовок `X-Real-IP` с IP клиента.

Аналогия: `upstream` — как список поваров, `ip_hash` — выбор повара по имени клиента, `proxy_pass` — передача заказа на кухню, `proxy_set_header` — уточнение деталей заказа.

Шаги:

1. Сохраните в `/etc/nginx/sites-available/app`.
2. Активируйте:

```
    sudo ln -s /etc/nginx/sites-available/app /etc/nginx/sites-enabled/

```

3. Проверьте:

```
    sudo nginx -t

```

4. Перезагрузите:

```
    sudo systemctl reload nginx

```

8 Ограничение ресурсов

Nginx ограничивает запросы и соединения для защиты.

8.1 Ограничение скорости запросов

```
http {
  limit_req_zone $binary_remote_addr zone=mylimit:10m rate=5r/s;
  server {
    listen 80;
    server_name example.com;
    location /api {
      limit_req zone=mylimit burst=10 nodelay;
      proxy_pass http://backend;
    }
  }
}
```

8.2 Подробный анализ

- `limit_req_zone $binary_remote_addr zone=mylimit:10m rate=5r/s;`:
  - **Что делает?** Создаёт зону памяти для ограничения запросов по IP.
  - **Зачем?** Защищает от DDoS-атак и перегрузки API.
  - **Как работает?** Хранит IP клиентов в зоне `mylimit` (10 МБ, 160k IP), ограничивая 5 запросов в секунду на IP.
  - **Ошибки**: Маленький размер зоны вызовет сброс старых записей.
- `limit_req zone=mylimit burst=10 nodelay;`:
  - **Что делает?** Применяет ограничение к `/api`, позволяя очередь запросов.
  - **Зачем?** Смягчает жёсткие лимиты, давая буфер для пиков.
  - **Как работает?** Если запросов больше 5 в секунду, до 10 лишних ставятся в очередь и обрабатываются сразу (`nodelay`).
  - **Ошибки**: Без `nodelay` очередь вызывает задержки.

Аналогия: `limit_req_zone` — как список посетителей клуба, `limit_req` — охранник, пропускающий 5 человек в секунду, а `burst` — очередь у входа.

Шаги:

1. Сохраните в `/etc/nginx/sites-available/limits`.
2. Активируйте:

```
    sudo ln -s /etc/nginx/sites-available/limits /etc/nginx/sites-enabled/

```

3. Проверьте:

```
    sudo nginx -t

```

4. Перезагрузите:

```
    sudo systemctl reload nginx

```

9 SSL/TLS

HTTPS шифрует трафик для безопасности.

9.1 Пример конфигурации

```
server {
  listen 80;
  server_name secure.example.com;
  return 301 https://$server_name$request_uri;
}
server {
  listen 443 ssl;
  server_name secure.example.com;
  ssl_certificate /etc/letsencrypt/live/secure.example.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/secure.example.com/privkey.pem;
  root /var/www/secure.example.com/html;
  index index.html;
}
```

9.2 Подробный анализ

- `return 301 https://$server_name$request_uri;`:
  - **Что делает?** Перенаправляет HTTP-запросы на HTTPS.
  - **Зачем?** Гарантирует шифрование всех соединений.
  - **Как работает?** Возвращает код 301, перенаправляя на `https://secure.example.com` с сохранением пути (`$request_uri`).
  - **Ошибки**: Неправильный `$server_name` сломает редирект.
- `listen 443 ssl;`:
  - **Что делает?** Прослушивает порт 443 с поддержкой SSL.
  - **Зачем?** Активирует HTTPS.
  - **Как работает?** Nginx использует SSL/TLS для шифрования соединений на порту 443.
  - **Ошибки**: Без `ssl` HTTPS не включится.
- `ssl_certificate`, `ssl_certificate_key`:
  - **Что делает?** Указывает пути к сертификату и ключу.
  - **Зачем?** Аутентифицирует сервер и шифрует данные.
  - **Как работает?** Nginx использует файлы для установки TLS-соединения.

– **Ошибки**: Неправильные пути или истёкший сертификат вызовут ошибку.

Аналогия: HTTPS — как замок на двери, `return 301` — указатель «вход через защищённую дверь», `ssl_certificate` — ключ от замка.

Шаги:

1. Установите Certbot:

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

```

2. Получите сертификат:

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

```

3. Сохраните в `/etc/nginx/sites-available/secure`.

4. Активируйте:

```
    sudo ln -s /etc/nginx/sites-available/secure /etc/nginx/sites-enabled/

```

5. Проверьте:

```
    sudo nginx -t

```

6. Перезагрузите:

```
    sudo systemctl reload nginx

```

10 Логирование

Логи помогают анализировать трафик.

10.1 Пример конфигурации

```
http {
  log_format custom '$remote_addr - $remote_user [$time_local] '
                    '"$request" $status $body_bytes_sent '
                    '"$http_referer" "$http_user_agent"';
  access_log /var/log/nginx/access.log custom;
  server {
    listen 80;
    server_name example.com;
    root /var/www/example.com/html;
  }
}
```

10.2 Подробный анализ

- `log_format custom '...';`:
  - **Что делает?** Определяет формат логов.
  - **Зачем?** Делает логи читаемыми и полезными.
  - **Как работает?** Задаёт шаблон с переменными (`$remote_addr` — IP, `$request` — запрос, `$status` — код ответа).
  - **Ошибки**: Неправильные переменные сломают формат.
- `access_log /var/log/nginx/access.log custom;`:
  - **Что делает?** Записывает логи в файл с указанным форматом.
  - **Зачем?** Сохраняет историю запросов.
  - **Как работает?** Каждый запрос записывается в `/var/log/nginx/access.log` в формате `custom`.
  - **Ошибки**: Неправильный путь или отсутствие прав вызовут ошибку.

Аналогия: `log_format` — как шаблон дневника, `access_log` — сам дневник, куда записываются события.

Шаги:

1. Отредактируйте `/etc/nginx/nginx.conf`.
2. Проверьте:

```
    sudo nginx -t

```

3. Перезагрузите:

```
    sudo systemctl reload nginx

```

11 Интеграция с Docker

Nginx в Docker упрощает развёртывание.

11.1 Пример конфигурации

```
version: "3.8"
services:
  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./html:/var/www/html
    networks:
      - app-network
  app:
    image: node:16
    working_dir: /app
    volumes:
      - ./app:/app
```

```
    command: ["node", "server.js"]
    networks:
      - app-network
networks:
  app-network:
    driver: bridge
```

```
http {
  server {
    listen 80;
    server_name localhost;
    location / {
      root /var/www/html;
      try_files $uri $uri/ /index.html;
    }
    location /api {
      proxy_pass http://app:3000;
    }
  }
}
```

11.2 Подробный анализ

- `location /api { proxy_pass http://app:3000; }`:
  - **Что делает?** Проксирует запросы `/api` к контейнеру `app`.
  - **Зачем?** Связывает Nginx с приложением в Docker.
  - **Как работает?** Nginx перенаправляет запросы к сервису `app` (разрешённому через сеть `app-network`) на порт 3000.
  - **Ошибки**: Неправильное имя сервиса или порт вызовут 502.

Аналогия: Nginx в Docker — как швейцар в отеле, `proxy_pass` — звонок в номер гостя.

Шаги:

1. Создайте `docker-compose.yml` и `nginx.conf`.
2. Запустите:

```
    docker-compose up -d

```

12 Команды Nginx

nginx -t Проверяет синтаксис:

```
    sudo nginx -t

```

13 Полезные советы

- Проверяйте синтаксис: `sudo nginx -t`.
- Изучайте https://nginx.org.