



# Web-технологии

Введение

# Состав курса

- Лекции
  - Курс на [moevm.info](http://e.moevm.info) «Web-технологии»  
<http://e.moevm.info/course/view.php?id=17>
  - Гоше Х.Д. «HTML5 для профессионалов»
- Лабораторные работы – **до 30 ноя.**
  - Беляев С.А. «Web-технологии. Лабораторный практикум», 2019 (библиотека)
- Курсовая работа – **до 30 ноя.**
  - Беляев С.А. «Разработка игр на языке JavaScript» (библиотека)
    - Изд-во: Лань, 2020 г.
    - <https://e.lanbook.com/book/138172>
  - Отдельное задание
- Три контрольные точки КТ-1, КТ-2 и КТ-3 в [ves.etu.ru](http://ves.etu.ru)
- **При нарушении сроков – снижение оценки за курс!**
- **Ловлю на плагиате – сразу «НЕУД.»**
- E-mail: [sabeliaev@etu.ru](mailto:sabeliaev@etu.ru)

- Все делают в соответствии с общим заданием в соответствии с учебным пособием на «**ЧИСТОМ**» JavaScript (**ES6**). В группах по 1 человеку.

1. Минимум 2 уровня игры
2. Реализованы все менеджеры в соответствии с учебным пособием (УП)
3. Есть таблица рекордов
4. Есть препятствия
5. Есть «интеллектуальные» противники и «бонусы»
6. Используются tiles с редактором Tiled ([www.mapeditor.org](http://www.mapeditor.org)) в соответствии с УП

- НО для владеющих **VUE.JS/NODE.JS** есть возможность выполнить отдельный набор задач (до **15 сен.** подойти и спросить задание)

# Содержание курса «Web-технологии»<sup>4</sup>

- HTTP-протокол, web-сервер (nginx, apache)
- Основы языка JavaScript
- Основы Node.js, основы Deno
- Основы HTML и CSS (LESS, SASS)
- Разработка сервера: Express, Koa, NestJS
- Шаблоны HTML-страниц: PUG, EJS
- RESTful, Ajax
- Основы использования jQuery, jQuery UI
- Основы языка TypeScript, применение статического анализа JavaScript
- Разработка с использованием фреймворка Angular
- Использование библиотеки React
- Разработка с использованием фреймворка Vue
- Модульное тестирование, e2e-тестирование, TDD
- Сборка приложений с использованием GULP и Webpack
- PHP, безопасность web-приложений

- Введение
- Трехуровневая архитектура
- Протокол HTTP
  - методы
  - параметры
  - запрос, ответ
  - заголовки
  - общие вопросы безопасности
- nginx
  - установка
  - конфигурации
  - статическое содержимое
  - использование в качестве прокси
- Apache HTTP-сервер

# Преимущества web-приложений

Простой доступ к  
информации

Простая  
поддержка,  
меньше стоимость  
распространения

Независимость от  
платформы

Широкая  
доступность

# Клиент-серверная модель

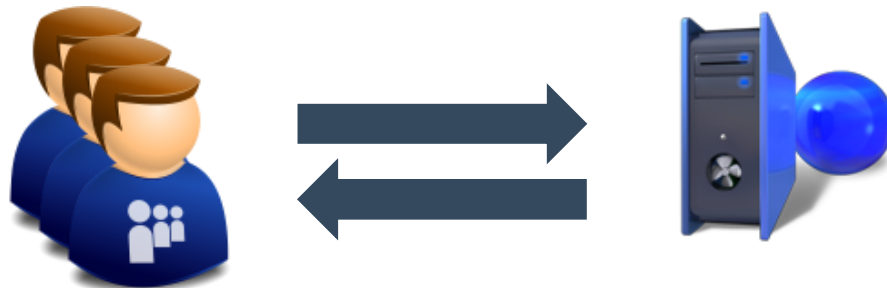
7



Настольное приложение

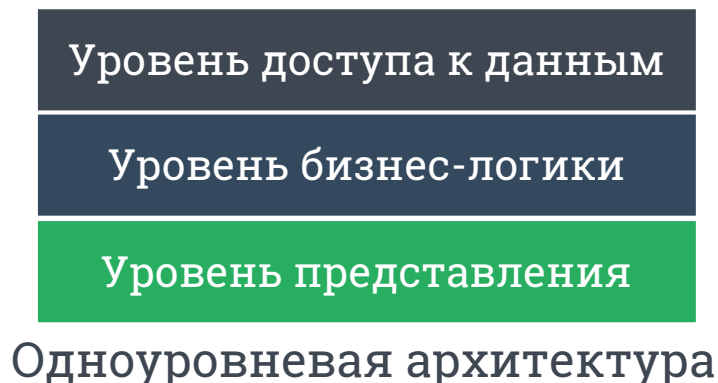


Сетевое приложение



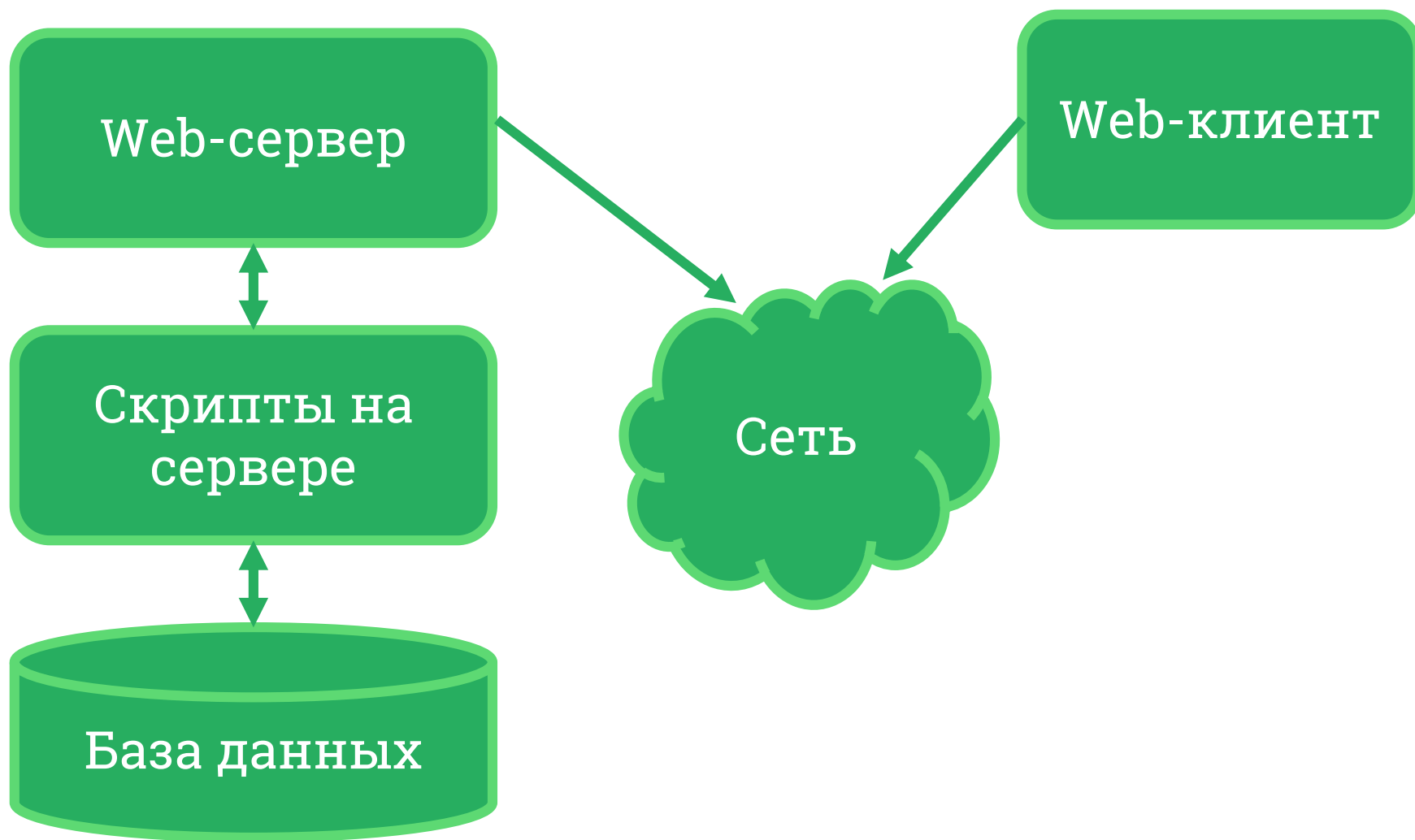
Web-приложение

# Архитектура web-приложений

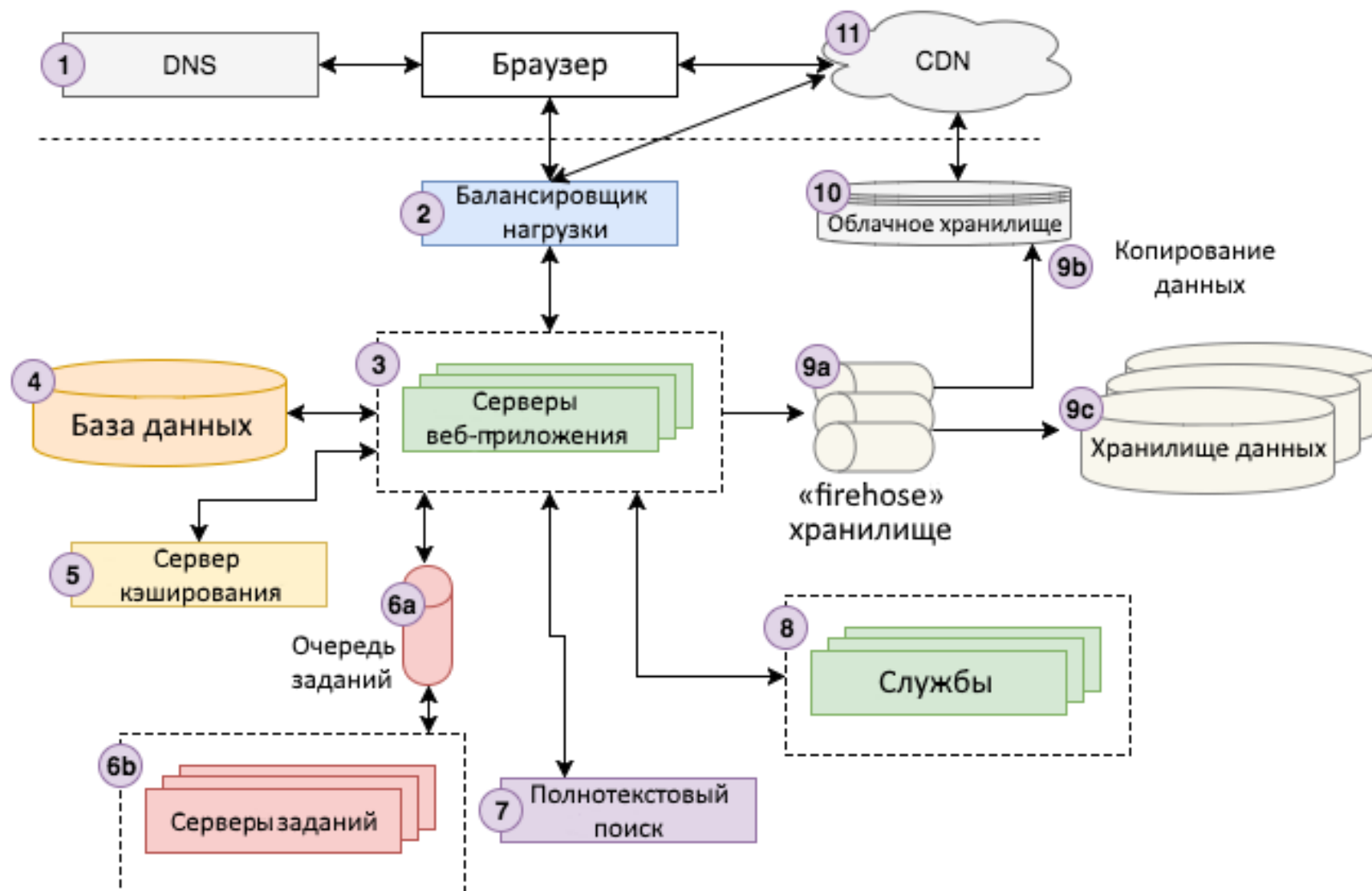




# Базовая архитектура web-приложения



# Архитектура современного web-приложения



# Протоколы прикладного уровня

- **HyperText Transfer Protocol – HTTP**
- **HyperText Transfer Protocol Secure – HTTPS**
- **File Transfer Protocol – FTP**
- **Internet Message Access Protocol – IMAP**
- **Post Office Protocol Version 3 – POP3**
- **Simple Network Management Protocol – SNMP**
- **Real Time Streaming Protocol – RTSP**
- ...

# Основные особенности HTTP

12

- Простой
  - HTTP-сообщения могут читаться и пониматься людьми
- Расширяемый
  - HTTP-заголовки сделали протокол лёгким для расширения и экспериментирования
- Не имеет состояния, но имеет сессию
  - Не существует связи между двумя запросами
  - Cookies позволяют использовать сессии с сохранением состояния

- HTTP/0.9
  - упорядочены правила взаимодействия клиента и сервера
- HTTP/1.0
  - открывал TCP-соединение для каждого обмена запросом/ответом
- HTTP/1.1 – 1999 г.
  - предоставил конвейерную обработку и устойчивые соединения (частично контролируется через заголовок Connection)
- HTTP/2.0 – поддерживается большинством браузеров с 2016г.
  - мультиплексирование сообщений («тёплые» соединения)
  - бинарный
  - сжатие передаваемых заголовков
  - явная приоритезация запросов
- HTTP/3 (HTTP-over-QUIC)

# Общие функции, управляемые с HTTP

- Кеш
  - Что и как долго кешировать
- Ослабление ограничений источника
  - Для предотвращения шпионских и других, нарушающих приватность, вторжений, веб-браузер обеспечивает строгое разделение между веб-сайтами
- Аутентификация
  - Некоторые страницы доступны только специальным пользователям
- Прокси и туннелирование
  - Серверы и/или клиенты часто располагаются в интернете, и скрывают свои истинные IP-адреса от других. HTTP запросы идут через прокси для пересечения этого сетевого барьера.
- Сессии
  - Использование HTTP cookies позволяет связать запрос с состоянием на сервере. Это создаёт сессию, хотя ядро HTTP -- протокол без состояния.

# Uniform Resource Identifier – URI

15

- URI является либо URL, либо URN, либо одновременно обоими
  - URL – местонахождение ресурса
  - URN – идентификация ресурса
- URI = [ схема ":" ] иерархическая-часть [ "?" запрос ] [ "#" фрагмент ]
- regexp: `^(([^:/?#]+):)?(//([^/?#]*))?([^?#]*(\?([^#]*)?)(#.*)?)?`
- Обработка кириллицы (Юникод):
  - <https://ru.wikipedia.org/wiki/Кириллица>
  - <https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D1%80%D0%B8%D0%BB%D0%BB%D0%B8%D1%86%D0%B0>
- Поддерживаются как абсолютные, так и относительные URI
- Следующие URI эквивалентны:
  - <http://abc.com:80/~smith/home.html?id=15>
  - <http://ABC.com/%7Esmith/home.html?id=15>
  - <http://ABC.com:/%7esmith/home.html?id=15>

Здесь 80 – стандартный порт http протокола

1. Открытие TCP соединения
  - открытие нового соединения
  - переиспользование существующего
  - открытие нескольких TCP-соединений к серверу
2. Отправка HTTP-сообщения
3. Чтение ответа от сервера
4. Заккрытие или переиспользование соединения для дальнейших запросов



# HTTP запросы и ответы

17

- Стартовая строка, описывающая запрос, или статус
  - Это всегда одна строка!
- Произвольный набор HTTP заголовков
- Пустая строка
- Произвольное тело

## Запрос

GET /wiki/страница HTTP/1.1  
Host: ru.wikipedia.org  
User-Agent: Mozilla/5.0 (X11; U;  
Linux i686; ru; rv:1.9b5)  
Gecko/2008050509 Firefox/4.0  
Accept: text/html  
Connection: close  
(пустая строка)

## Ответ

HTTP/1.1 200 OK  
Date: Wed, 11 Feb 2012 12:22:00 GMT  
Server: Apache  
X-Powered-By: PHP/5.2.4-2ubuntu5  
Last-Modified: Wed, 11 Feb 2012  
12:22:00 GMT  
Content-Language: ru  
Content-Type: text/html;  
charset=utf-8  
Content-Length: 3421  
Connection: close  
(пустая строка)  
(далее – запрошенная  
страница в HTML)

# Запросы HTTP. Стартовая строка

18

- Метод HTTP
  - глагол (например, GET, PUT или POST)
  - существительное (например, HEAD или OPTIONS)
- Цель запроса, обычно URL. Формат цели запроса:
  - Абсолютный путь, за которым следует '?' и строка запроса
    - POST / HTTP 1.1
    - GET /background.png HTTP/1.0
    - HEAD /test.html?query=alibaba HTTP/1.1
    - OPTIONS /anypage.html HTTP/1.0
  - Полный URL
    - GET http://developer.mozilla.org/en-US/docs/Web/HTTP/Messages HTTP/1.1
- Версия HTTP, определяющая структуру оставшегося сообщения

# Запросы HTTP. Методы

- **GET**
  - запрашивает представление ресурса
- **POST**
  - используется для отправки сущностей к определённому ресурсу
- **PUT**
  - заменяет все текущие представления ресурса данными запроса
- **DELETE**
  - удаляет указанный ресурс
- **HEAD**
  - запрашивает ресурс так же, как и метод GET, но без тела ответа
- **CONNECT**
  - устанавливает "туннель" к серверу, определённому по ресурсу
- **OPTIONS**
  - используется для описания параметров соединения с ресурсом
- **TRACE**
  - выполняет вызов возвращаемого тестового сообщения с ресурса
- **PATCH**
  - используется для частичного изменения ресурса

# Запросы HTTP. Заголовки

- Основные заголовки – General Headers
  - могут включаться в любое сообщение клиента и сервера
- Заголовки запроса – Request Headers
  - используются только в запросах клиента
  - например, Referer, User-Agent
- Заголовки ответа – Response Headers
  - только для ответов от сервера
  - например, Allow
- Заголовки сущности – Entity Headers
  - сопровождают каждую сущность сообщения
  - например, Content-Language

# Заголовки. Кэширование

- `max-age=[секунды]`
  - описывает максимальный период времени, в течение которого контент остается свежим
- `no-cache`
  - принуждает кэш отправлять запрос на исходный сервер каждый раз для валидации
- `no-store`
  - указывает кэшу не сохранять копию контента, ни при каких условиях
- Примеры
  - `Cache-Control: max-age=3000`
  - `Cache-Control: no-cache`
  - `Cache-Control: no-store`

# Media Types

- media-type = type "/" subtype \*( ";" parameter )

- Пример

- Асепт: image/gif

application

audio

example

image

message

model

multipart

text

video

## Изображение

- image/gif
- image/jpeg
- image/pjpeg
- image/png
- image/svg+xml
- image/tiff
- image/vnd.microsoft.icon
- image/vnd.wap.wbmp
- image/webp

**MIME-типы**  
**(Multipurpose**  
**Internet Mail**  
**Extension)**

# Ответы HTTP. Строка статуса

- Версию протокола, обычно HTTP/1.1 или HTTP/2.0
- Код состояния (status code), показывающая, был ли запрос успешным
  - Примеры: 200, 404 или 302
- Пояснение (status text)
  - Краткое текстовое описание кода состояния, помогающее пользователю понять сообщение HTTP
- Пример строки статуса
  - HTTP/1.1 404 Not Found

# Коды состояний HTTP

- 1xx: Informational (информационные)
- 2xx: Success (успешно):
  - 200 OK («хорошо»)
- 3xx: Redirection (перенаправление)
- 4xx: Client Error (ошибка клиента):
  - 403 Forbidden («запрещено»)
  - 404 Not Found («не найдено»)
  - 405 Method Not Allowed («метод не поддерживается»)
- 5xx: Server Error (ошибка сервера):
  - 500 Internal Server Error («внутренняя ошибка сервера»)
- Пример  
HTTP/1.1 200 Ok  
Date: Tue, 26 Dec 2017 17:33:13 GMT  
Content-Type: text/html; charset=UTF-8  
Cache-Control: no-cache, no-store, max-age=0,  
must-revalidate  
Expires: Tue, 26 Dec 2017 17:33:13 GMT  
Last-Modified: Tue, 26 Dec 2017 17:33:13 GMT



# Пример скачивания файла

25

- **Пример запроса**

- GET /**movie.avi** HTTP/1.0
- Host: example.org
- Accept: \*/\*
- User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)
- Referer: http://example.org/

- **Пример ответа**

- HTTP/1.1 200 OK
- Date: Sat Dec 30 2017 20:17:47 GMT
- Server: Apache/2.2.3
- Last-Modified: Sat Dec 30 2017 20:17:47 GMT
- ETag: "686897696a7c876b7e"
- Content-Type: video/x-msvideo
- **Content-Length: 160993792**
- Accept-Ranges: bytes
- Connection: close
- (пустая строка)
- (двоичное содержимое всего файла)

# Пример фрагментарного скачивания файла

## • Пример запроса

- GET /movie.avi HTTP/1.0
- Host: example.org
- Accept: \*/\*
- User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)
- Range: bytes=37748736-
- Referer: http://example.org/

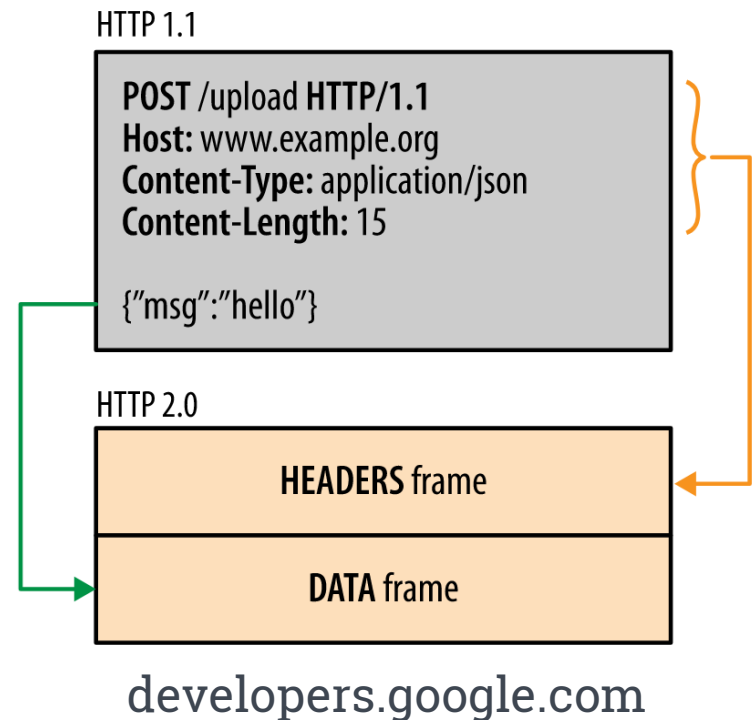
## • Пример ответа

- HTTP/1.1 206 Partial Content
- Date: Sat Dec 30 2017 20:23:57 GMT
- Server: Apache/2.2.3
- Last-Modified: Sat Dec 30 2017 20:23:57 GMT
- ETag: "686897696a7c876b7e"
- Accept-Ranges: bytes
- **Content-Range: bytes 37748736-88080384/88080385**
- **Content-Length: 50331649**
- Content-Type: video/x-msvideo
- Connection: close
- (пустая строка)
- (двоичное содержимое от 36-го мегабайта)

# HTTP/2 (1)

27

- Совместимость с концепциями HTTP 1.1
- Уменьшение задержек доступа для ускорения загрузки страниц:
  - Сжатие данных в заголовках HTTP
  - Использование push-технологий на серверной стороне
  - Конвейеризация запросов
  - Обеспечение множества запросов в одном соединении TCP
- Сохранение совместимости с веб-браузерами

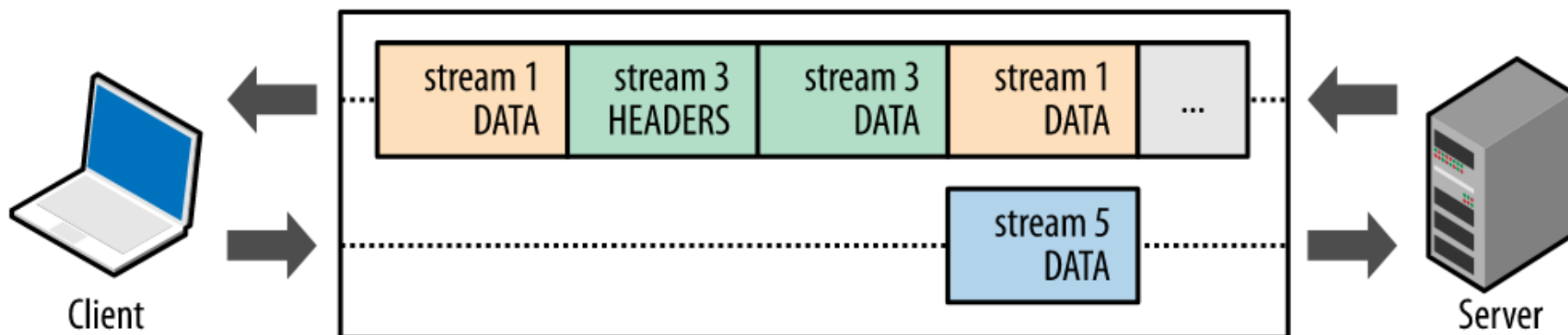


# HTTP/2 (2)

- Вся связь осуществляется через одно TCP-соединение
- Каждый поток имеет уникальный идентификатор и необязательную информацию о приоритете
- Каждое сообщение является логическим HTTP-сообщением
  - состоит из одного или нескольких кадров
- Кадр – это наименьшая единица связи, которая переносит данные определенного типа – например
  - заголовки HTTP
  - полезную нагрузку сообщения...
- Кадры из разных потоков могут чередоваться

# Преимущества HTTP/2

- Чередование нескольких запросов параллельно, не блокируя ни один
- Чередование нескольких ответов параллельно, не блокируя ни один
- Использование одного соединения для одновременной доставки нескольких запросов и ответов
- Удаление обходных путей HTTP/1.x
- Снижение времени загрузки страницы
- ...



# Ограничения HTTP/2

- Сложная реализация server push
- Блокировка запросов на уровне TCP
- Параллельные запросы увеличивают нагрузку на сервер – возможен таймаут запросов
- В случае медленного соединения все соединения «сбросятся» до одного HTTP/2

- Основные сходства с HTTP/2
  - Активный сервер
  - Мультиплексирование через 1 соединение с потоками
  - Приоритеты ресурсов
  - Сжатие заголовков
- Основные отличия от HTTP/2
  - Использование QUIC вместо TCP
    - QUIC – экспериментальный интернет протокол от Google
  - Быстрое установление безопасного соединения
  - Не поддерживает нешифрованное соединение

# nginx – web-сервер для высоконагруженных сайтов

Яндекс

Mail.Ru

ВКонтакте

Рамблер

Согласно статистике Netcraft nginx обслуживал или проксировал **23.59%** из общего списка сайтов в августе 2023 года

<https://news.netcraft.com/archives/category/web-server-survey/>

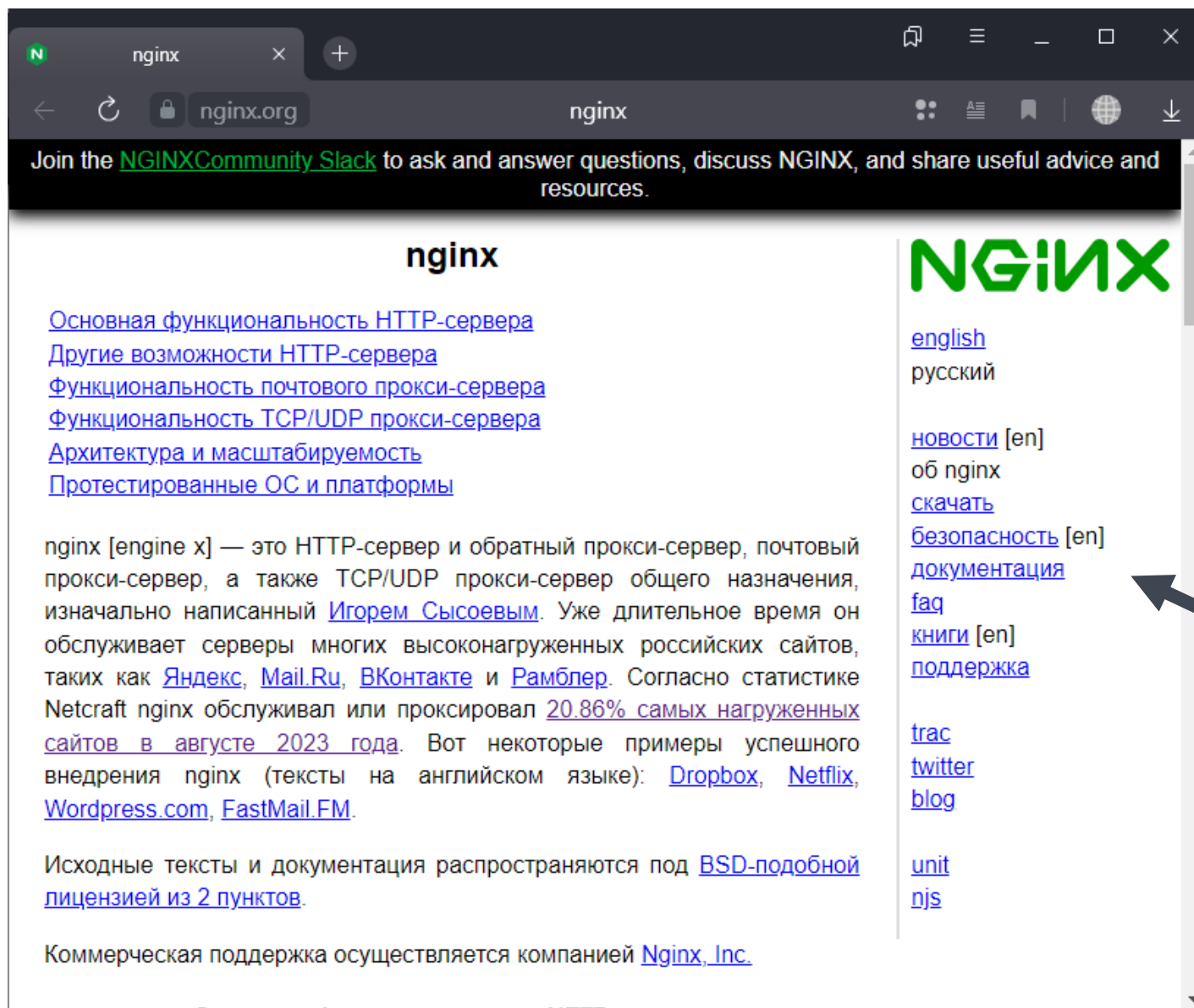
<http://nginx.org/ru/>

<http://help.ubuntu.ru/wiki/nginx-phpfpm>



# Основная функциональность HTTP-сервера nginx

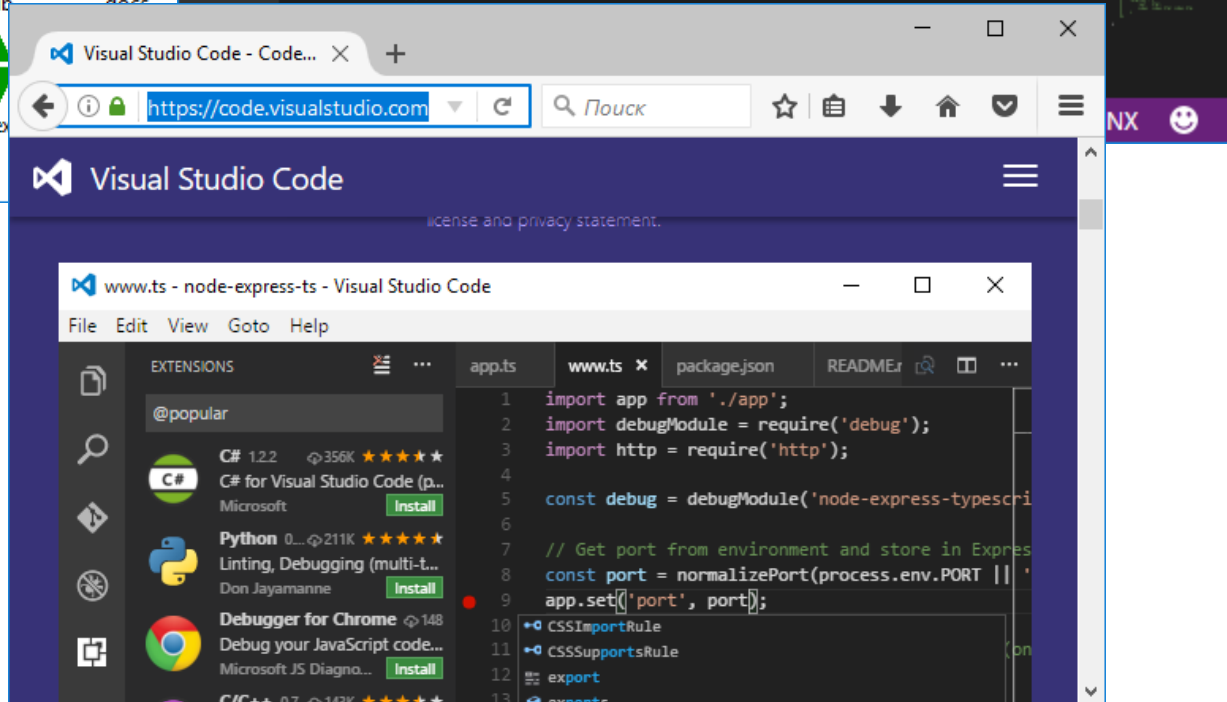
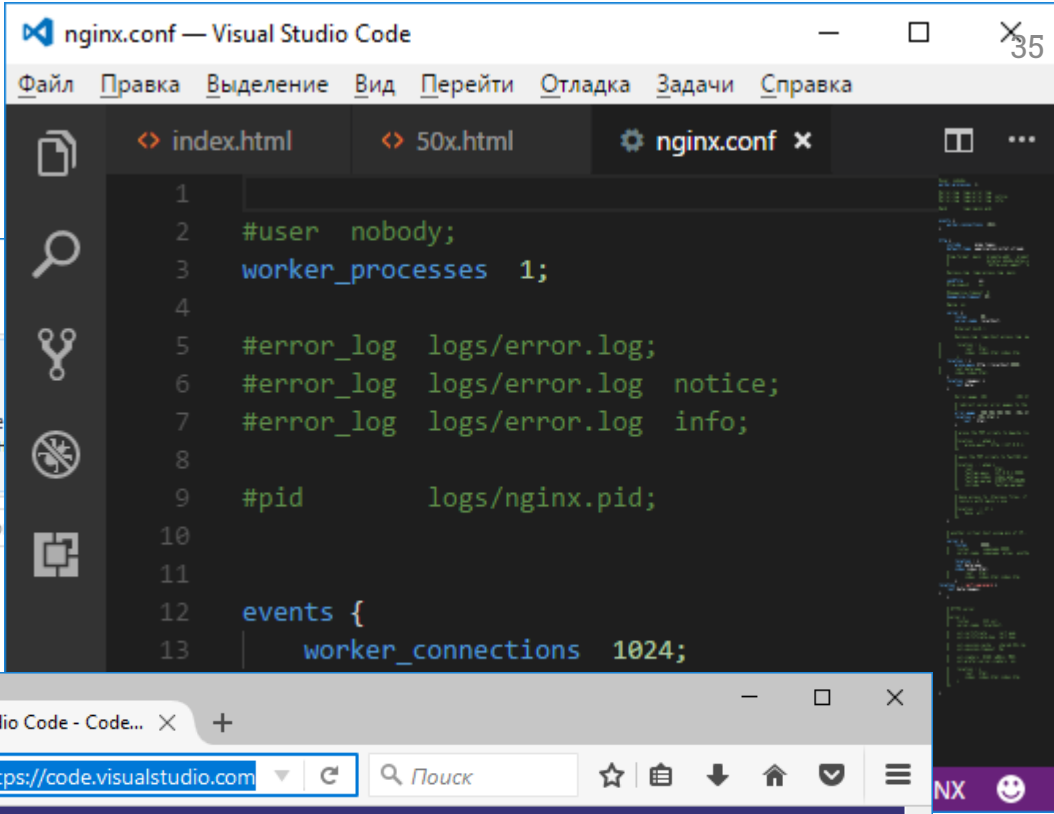
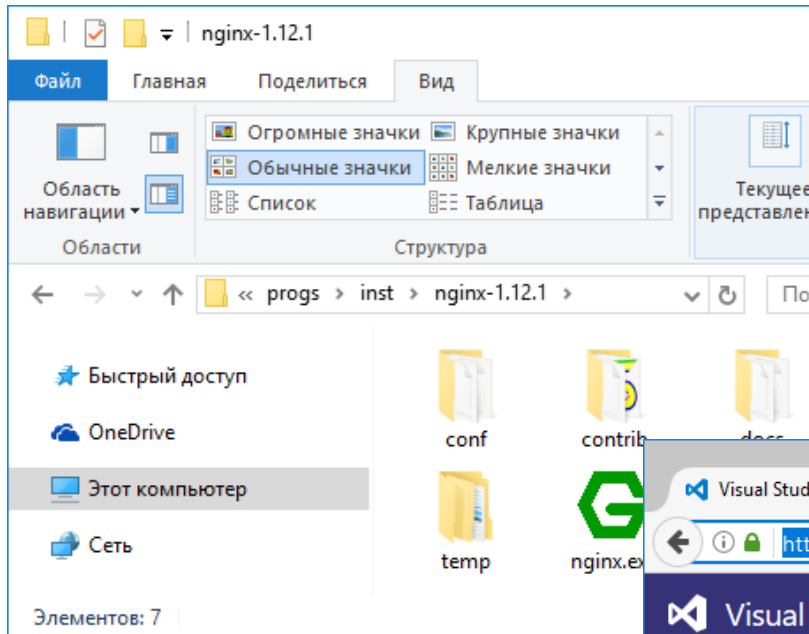
- Обслуживание статических запросов, индексных файлов, автоматическое создание списка файлов, кэш дескрипторов открытых файлов.
- Проксирование с кэшированием, распределение нагрузки и обеспечение отказоустойчивости.
- Модульность, фильтры, в том числе сжатие, преобразование изображений; несколько подзапросов на одной странице параллельно.
- Виртуальные серверы, определяемые по IP-адресу и имени.
- Поддержка keep-alive и pipelined соединений.
- Выполнение разных функций в зависимости от адреса клиента.
- Ограничение скорости отдачи ответов.
- Зеркалирование запросов.



<http://nginx.org/ru>  
<https://www.nginx.com/>

Часть функционала бесплатна, часть –  
 коммерческая подписка

# nginx



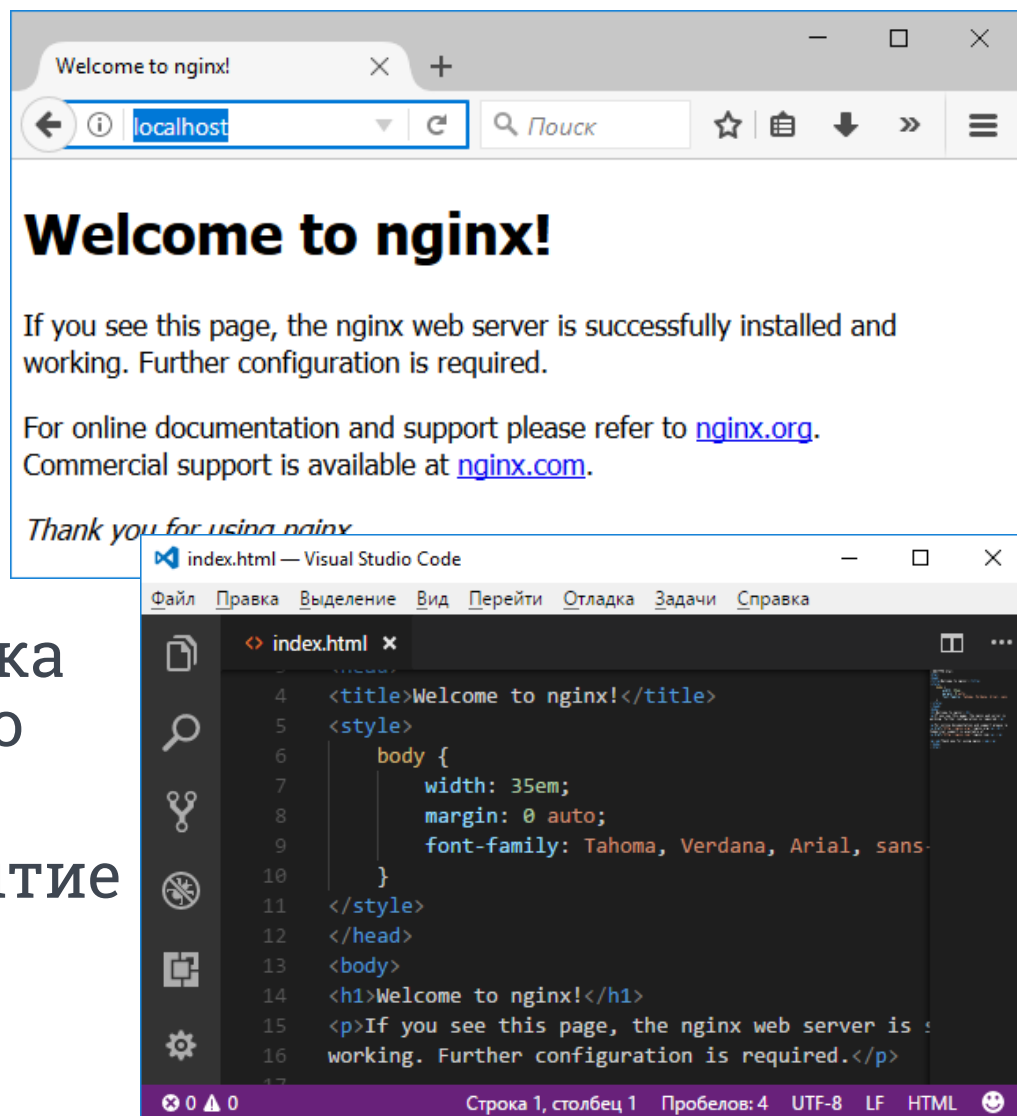
`sudo apt-get nginx`

`nginx -h`

# Команды nginx

36

- **nginx -s сигнал**
- Сигналы:
  - **stop** — быстрое завершение
  - **quit** — плавное завершение
  - **reload** — перезагрузка конфигурационного файла
  - **reopen** — переоткрытие лог-файлов
- Пример:
  - **nginx -s quit**

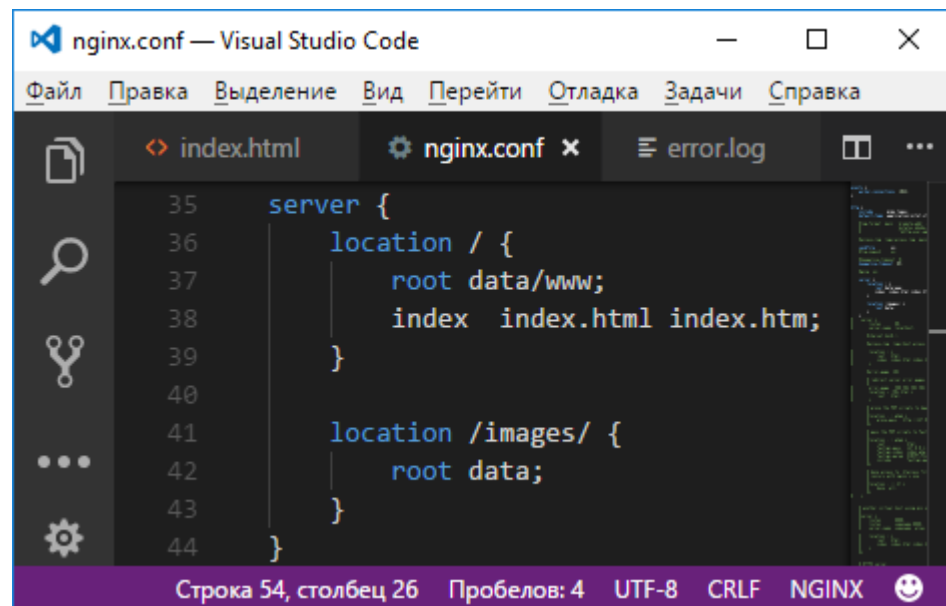


# Раздача статического содержимого 37

- **nginx/conf/nginx.conf**
- **/etc/nginx/sites-available/default**

```
server {  
    location / {  
        root /data/www;  
    }  
    location /images/ {  
        root /data;  
    }  
}
```

<http://help.ubuntu.ru/wiki/nginx-phpfpm>

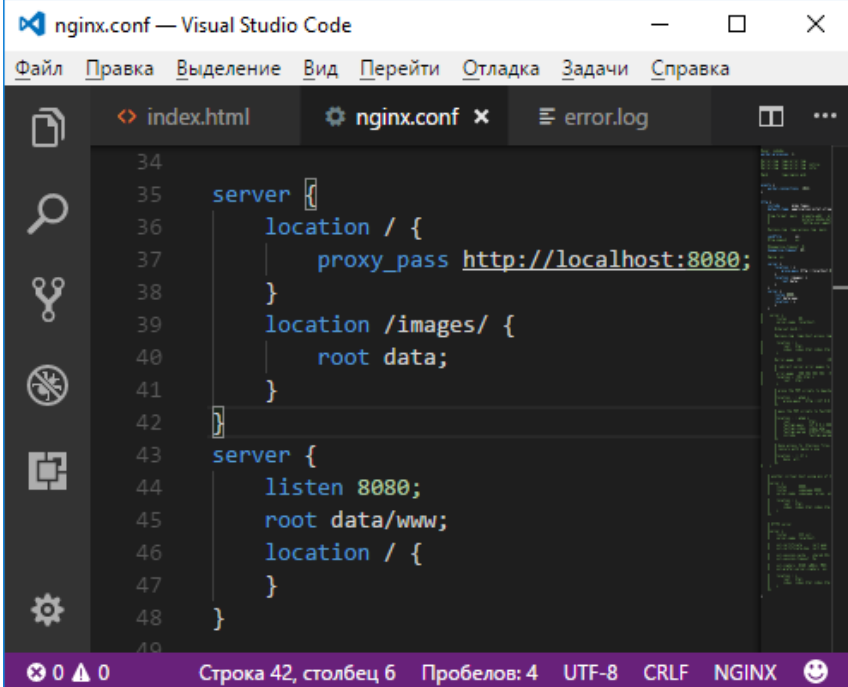


## Две папки:

- для «корня»
- для «изображений»

# Настройка простого прокси-сервера <sup>38</sup>

```
server {  
    listen 8080;  
    root data/www;  
    location / {  
        proxy_pass http://ubuntu.ru;  
    }  
}
```



The screenshot shows the Visual Studio Code editor with the file 'nginx.conf' open. The configuration is as follows:

```
34  
35 server {  
36     location / {  
37         proxy_pass http://localhost:8080;  
38     }  
39     location /images/ {  
40         root data;  
41     }  
42 }  
43 server {  
44     listen 8080;  
45     root data/www;  
46     location / {  
47     }  
48 }
```

The status bar at the bottom indicates 'Строка 42, столбец 6' (Line 42, Column 6), 'Пробелов: 4' (Spaces: 4), 'UTF-8', 'CRLF', 'NGINX', and a smiley face icon.

Порт 8080

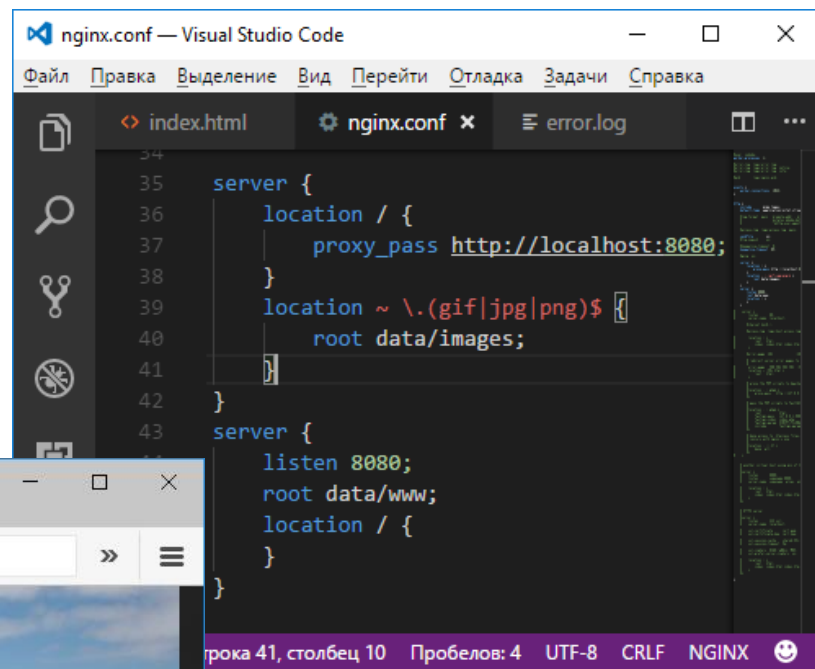
В данном примере запрос на 8080 будет передан на адрес <http://ubuntu.ru>

# Регулярные выражения

39

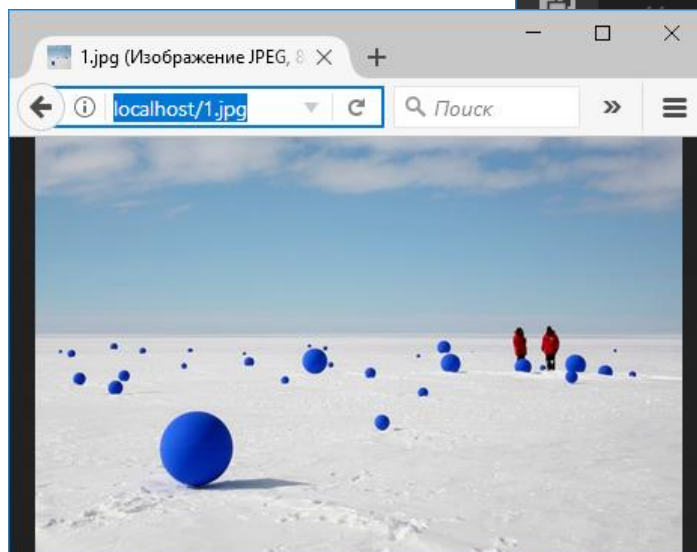
```
server {  
    location / {  
        proxy_pass http://localhost:8080;  
    }  
    location ~ \.(gif|jpg|png)$ {  
        root data/images;  
    }  
}
```

```
server {  
    listen 8080;  
    root data/www;  
    location / {  
    }  
}
```



```
nginx.conf — Visual Studio Code  
Файл Правка Выделение Вид Перейти Отладка Задачи Справка  
index.html nginx.conf x error.log  
34  
35 server {  
36     location / {  
37         proxy_pass http://localhost:8080;  
38     }  
39     location ~ \.(gif|jpg|png)$ {  
40         root data/images;  
41     }  
42 }  
43 server {  
44     listen 8080;  
45     root data/www;  
46     location / {  
47     }  
48 }
```

строка 41, столбец 10 Пробелов: 4 UTF-8 CRLF NGINX



# Настройка HTTPS-сервера (1)

Генерация «самоподписанного» сертификата

- **openssl req -new -sha256 -key example.com\_nginx.key -out example.com.csr**
- **cp example.com\_nginx.key /etc/ssl/private/**
- **chown www-data:www-data /etc/ssl/private/example.com\_nginx.key**
- **chmod 400 /etc/ssl/private/example.com\_nginx.key**
- **cp example.com\_nginx.crt /etc/ssl/certs/**



<https://letsencrypt.org/ru/> - бесплатные сертификаты для web-сервера



# Настройка HTTPS-сервера (2)

41

```
listen 443 ssl default_server;  
ssl_certificate /etc/ssl/certs/example.com_nginx.crt; # сертификат (можно  
свободно распространять)  
ssl_certificate_key /etc/ssl/private/example.com_nginx.key; # приватный ключ  
(секретный файл - НИКОМУ НЕ ПОКАЗЫВАТЬ)  
ssl_session_timeout 20m; # время 20 минут  
ssl_session_cache shared:SSL:20m; # размер кеша 20МБ  
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;  
ssl_prefer_server_ciphers on;  
ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-  
SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-  
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128-GCM-  
SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-  
AES128-SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-  
SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-  
RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-  
SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-  
SHA256:DHE-DSS-AES256-SHA:DHE-RSA-AES256-  
SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!3DES:!MD5:!PSK;
```

# Включение http2

42

```
server {  
    listen 443 ssl http2;  
  
    ssl_certificate example.com_nginx.crt;  
    ssl_certificate_key example.com_nginx.key;  
}
```

HTTP/2 работает только с SSL

## Традиционные порты HTTP:

- **80** – в браузере указывать не требуется (http://адрес.сайта)
- **8080** – обычно используется для разработки, но может быть и другой, например, 8081, 8090, 9080... (http://адрес.сайта:8080)

## Традиционные порты HTTPS:

- **443** – в браузере указывать не требуется (https://адрес.сайта)
- **8443** – обычно используется для разработки, но может быть и другой

## Сервера:

- **3000, 5000** – обычно используется как серверная часть для web-приложений, но может быть и другой

# Балансировка нагрузки (upstream)

43

```
upstream backend {  
    # Вес сервера (по умолчанию =1), рассматривается как пропорция весов  
    server backend1.example.com:80 weight=5;  
    # Время и количество попыток, когда сервер будет считаться недоступен  
    server 127.0.0.1:80 max_fails=3 fail_timeout=30s;  
    # Резервный сервер, к нему обращаются, когда остальные недоступны  
    server backup1.example.com:80 backup;  
}  
  
server {  
    location / {  
        proxy_pass http://backend  
    }  
}
```

Порт может быть другой

В данном случае в качестве серверов могут выступать, например, скрывающиеся за nginx сервера Node.JS (тот же Express)

[http://nginx.org/ru/docs/stream/nginx\\_stream\\_upstream\\_module.html#upstream](http://nginx.org/ru/docs/stream/nginx_stream_upstream_module.html#upstream)

# Пример HTTP/3 для nginx

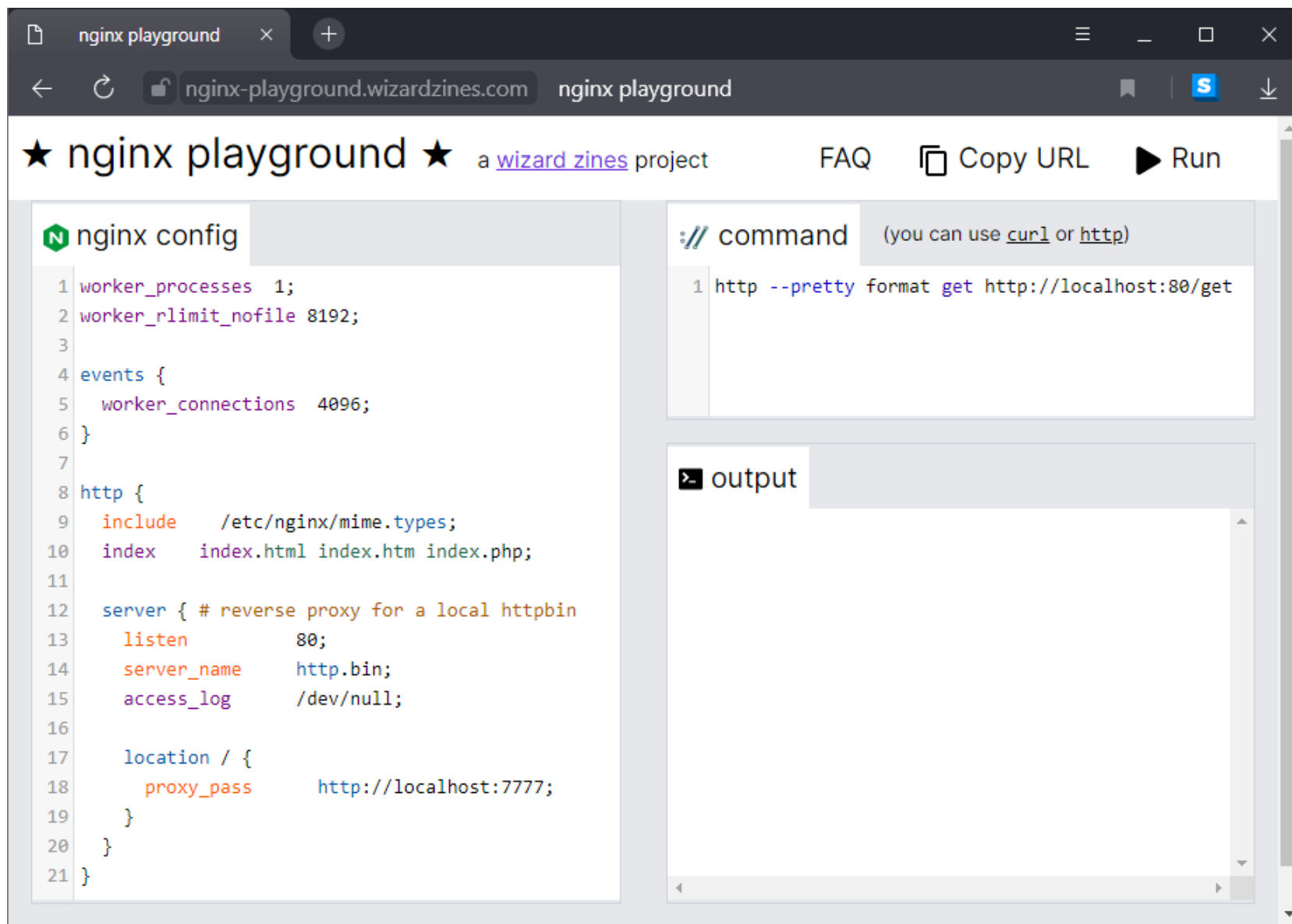
```
http {  
    log_format quic '$remote_addr - $remote_user [$time_local] '  
        '$request' $status $body_bytes_sent '  
        '$http_referer' '$http_user_agent' '$http3';  
  
    access_log logs/access.log quic;  
  
server {  
    # for better compatibility it's recommended  
    # to use the same port for quic and https  
    listen 8443 quic reuseport;  
    listen 8443 ssl;  
  
    ssl_certificate    certs/example.com.crt;  
    ssl_certificate_key certs/example.com.key;  
  
    location / {  
        # required for browsers to direct them to quic port  
        add_header Alt-Svc 'h3=":8443"; ma=86400';  
    }  
}  
}
```

# Фрагмент примера конфигурационного файла

```
1  user www-data;
2  worker_processes auto;
3  pid /run/nginx.pid;
4  include /etc/nginx/modules-enabled/*.conf;
5
6  events {
7      worker_connections 768;
8      multi_accept on; # Будет принимать максимально возможное количество соединений
9  }
10
11 http {
12     # Gazprom-classes settings
13     keepalive_requests 100; # Максимальное количество keepalive запросов от одного клиента
14     reset_timedout_connection on; # Если клиент перестал читать отвечать, Nginx будет сбрасывать соединение с ним
15     client_body_timeout 10; # Будет ждать 10 секунд тело запроса от клиента, после чего сбросит соединение
16     send_timeout 2; # Если клиент прекратит чтение ответа, Nginx подождет 2 секунды и сбросит соединение
17     ##
18     # Basic Settings
19     ##
20
21     sendfile on;
22     tcp_nopush on;
23     tcp_nodelay on;
24     keepalive_timeout 65;
25     types_hash_max_size 2048;
26     client_max_body_size 100M;
27     # server_tokens off;
28
29     server_names_hash_bucket_size 64;
30     # server_name_in_redirect off;
```

# «Песочница» nginx

46



The screenshot shows a web browser window with the address bar displaying 'nginx-playground.wizardzines.com'. The page title is 'nginx playground' and it is identified as 'a wizard zines project'. Navigation links include 'FAQ', 'Copy URL', and a 'Run' button. The main content area is divided into three sections: 'nginx config', 'command', and 'output'. The 'nginx config' section contains a pre-formatted nginx configuration file. The 'command' section has a text input field with a command. The 'output' section is an empty text area for the results of the command execution.

nginx playground ★ a [wizard zines](#) project [FAQ](#) [Copy URL](#) [Run](#)

**nginx config**

```
1 worker_processes 1;
2 worker_rlimit_nofile 8192;
3
4 events {
5     worker_connections 4096;
6 }
7
8 http {
9     include /etc/nginx/mime.types;
10    index index.html index.htm index.php;
11
12    server { # reverse proxy for a local httpbin
13        listen 80;
14        server_name http.bin;
15        access_log /dev/null;
16
17        location / {
18            proxy_pass http://localhost:7777;
19        }
20    }
21 }
```

**command** (you can use [curl](#) or [http](#))

```
1 http --pretty format get http://localhost:80/get
```

**output**

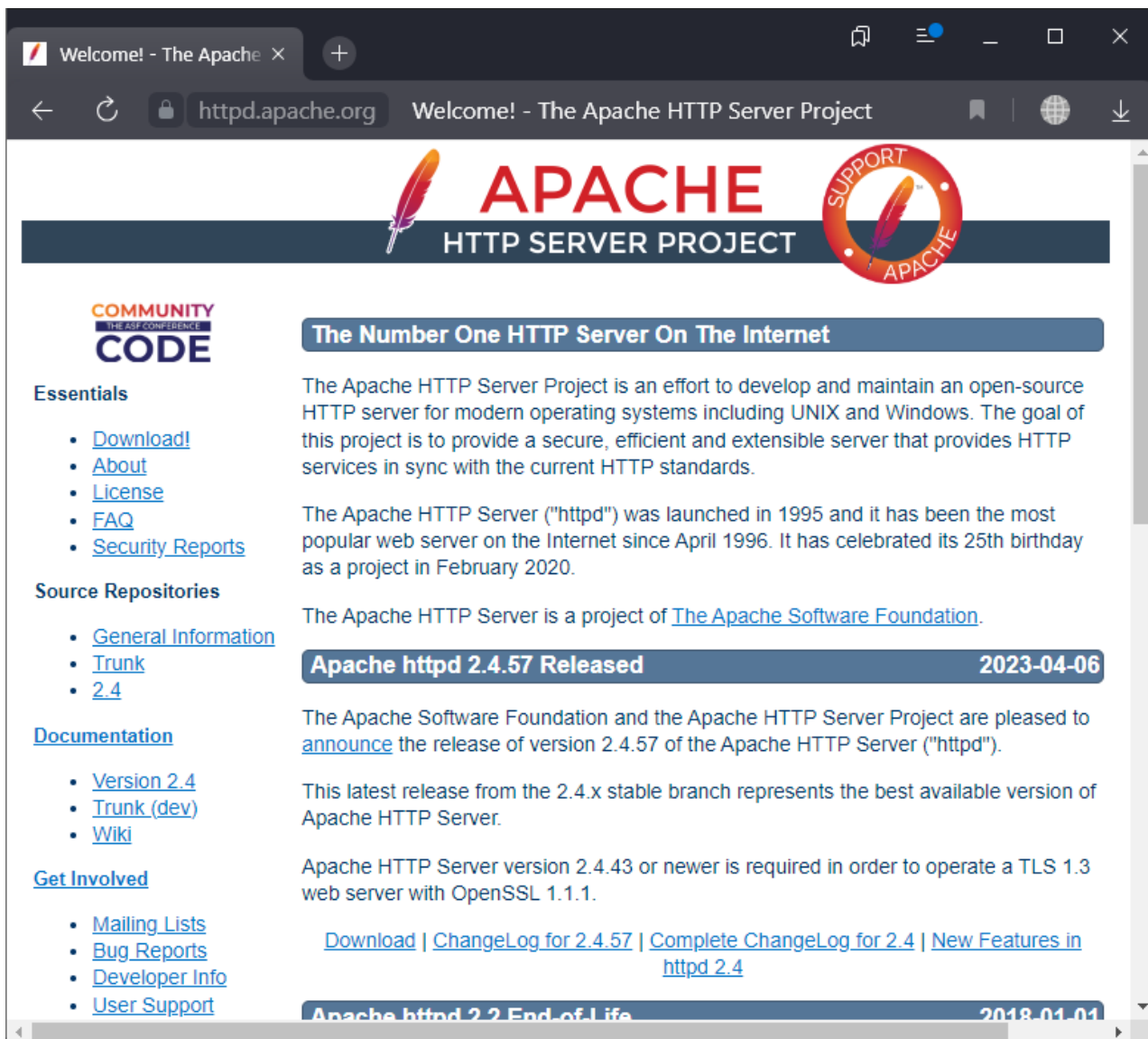
<https://nginx-playground.wizardzines.com/>

- **sudo apt-get install nginx**
- Конфигурационные файлы
  - **/etc/nginx/sites-available/default**
  - **nginx/conf/nginx.conf**
- Подключение конфигурационного файла
  - **include example.nginx;**
- Простейшая конфигурация

```
server {  
    listen 8080;  
    location / {  
        root /data/www;  
    }  
}
```

# Apache HTTP-сервер

48



Согласно статистике Netcraft Apache обслуживал или проксировал **20.81%** из общего списка сайтов в августе 2023 года

<http://help.ubuntu.ru/wiki/apache2>  
[https://ru.wikipedia.org/wiki/Apache\\_HTTP\\_Server](https://ru.wikipedia.org/wiki/Apache_HTTP_Server)



# Интеграция Apache с другим ПО и языками

- PHP (mod\_php)
- Python (mod python, mod wsgi)
- Ruby (apache-ruby)
- Perl (mod perl)
- ASP (apache-asp)
- Tcl (rivet)

# Сервер Apache

50

- **sudo apt-get install apache2**
- Конфигурационные файлы
  - **/etc/apache2/apache2.conf**
  - **/etc/apache2/sites-available/\*.conf**
- Настройка
  - **sudo mkdir -p /var/www/example.dev**
  - **sudo chmod -R 766 /var/www**
  - **nano /var/www/example.com/index.html**
  - **sudo touch /etc/apache2/sites-available/example.dev.conf**
  - **sudo nano /etc/apache2/sites-available/example.dev.conf**
- Конфигурация

```
<VirtualHost *:80>
    ServerName example.dev
    DocumentRoot /var/www/example.dev
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```
- Включение виртуального хоста
  - **sudo a2ensite example.dev**
  - **sudo service apache2 restart**

# Вопросы для самопроверки

51

- Понятны сроки сдачи л/р, к/р?
- Понятны требования к к/р?
- Какими функциями можно управлять с помощью HTTP?
- Какие отличия версий HTTP?
- Что такое HTTP-поток?
- Какие методы HTTP будут использоваться в л/р? Для чего они нужны?
- В чём особенности HTTP/2 и HTTP/3?
- Для чего нужны web-сервера? Их базовые возможности?
- Какую простейшую конфигурацию web-сервера можете описать?
- Как сделать простейший прокси-сервер?