

# Операционная система Linux: HTTP / HTTPS



Шило  
Петр



# Шило Петр

DevOps-инженер

Arifonica



[Шило Петр](#)

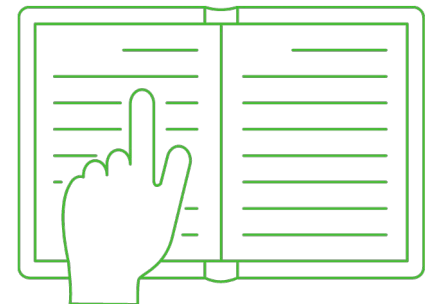
---

# Предисловие

На этом занятии мы поговорим о:

- важности HTTP;
- принципах работы HTTP;
- работе с HTTP на практике.

По итогу занятия вы получите представление о работе HTTP / HTTPS и настройке серверов для работы с ними.





# План занятия

1. [Предисловие](#)
2. [Структура протокола HTTP](#)
3. [Запуск и настройка HTTP сервера](#)
4. [S for security](#)
5. [HTTP2](#)
6. [RPC](#)
7. [Итоги](#)
8. [Домашнее задание](#)



# Структура протокола HTTP

# Обзор

**HTTP** (HyperText Transfer Protocol, протокол передачи гипертекста) – протокол изначально созданный для передачи документов HTML, но использующийся в данный момент для передачи любых данных.

В основе протокола лежит **технология клиент-сервер**.

## Пример запроса клиента

```
* Rebuilt URL to: netology.ru/  
* Trying 104.22.49.171...  
* TCP_NODELAY set  
* Connected to netology.ru (104.22.49.171) port 80 (#0)  
> GET / HTTP/1.1  
> Host: netology.ru  
> User-Agent: curl/7.61.1  
> Accept: */*  
>
```

## Пример ответа сервера

```
< HTTP/1.1 301 Moved Permanently  
< Date: Mon, 08 Mar 2021 10:08:20 GMT  
< Transfer-Encoding: chunked  
< Connection: keep-alive  
< Cache-Control: max-age=3600  
< Expires: Mon, 08 Mar 2021 11:08:20 GMT  
< Location: https://netology.ru/  
< cf-request-id: 08b2e834ad00004e742f078000000001  
< Server: cloudflare  
< CF-RAY: 62cb43011f144e74-FRA  
< alt-svc: h3-27=":443"; ma=86400, h3-28=":443"; ma=86400  
<  
* Connection #0 to host netology.ru left intact
```

# Структура протокола

**Стартовая строка** – содержит параметры типа сообщения, имеет различную структуру для клиента и сервера.

Имеет вид:

- Для клиента:

Метод URI HTTP/Версия

Пример: GET / HTTP/1.1

- Для сервера:

HTTP/Версия КодСостояния Пояснение

Пример: HTTP/1.1 200 ОК

---

# Структура протокола

**Заголовки** – используются для передачи любых дополнительных параметров.

→ Например, о типе данных в сообщении или о наличии.

Начиная с версии 1.1 заголовок **Host** является обязательным.

**Тело** – непосредственно сами данные.



# Методы HTTP запроса

Метод	Описание
GET	Используется для получения данных с сервера.
POST	Применяется при передаче данных от клиента на сервер.
PUT	Используется для загрузки данных на указанный ресурс.
OPTIONS	Получение информации о сервере. Обычно используется для получения возможностей сервера.
HEAD	Аналогично GET но без получения данных с сервера. Ранее использовался для получения метаданных но загрузки самих данных.
PATCH	Аналогичен PUT но используется для правки только части данных.
DELETE	Удаляет ресурс с сервера.

---

# Коды состояния HTTP

Код	Описание
1XX	Информационные ответы.
2XX	Успех. Сообщает об успехе того или иного запроса.
3XX	Перенаправление. Сообщает о том что ресурс доступен по другому адресу.
4XX	Сообщает о наличии ошибки со стороны клиента.
5XX	Сообщает о наличии ошибки на стороне сервера.

# MIME типы


В **header** передается **Content-type**.

➔ Позволяет клиенту понять содержимое ответа и по разному обрабатывать различный контент.

- **text/plain** – является типом по умолчанию для текстовых файлов;
- **application/octet-stream** – является типом по умолчанию для всех остальных случаев.

## Другие примеры:

- **text/html** – HTML содержимое;
- **image/jpeg** – JPEG изображения;
- **audio/mpeg** – mp3 аудио.



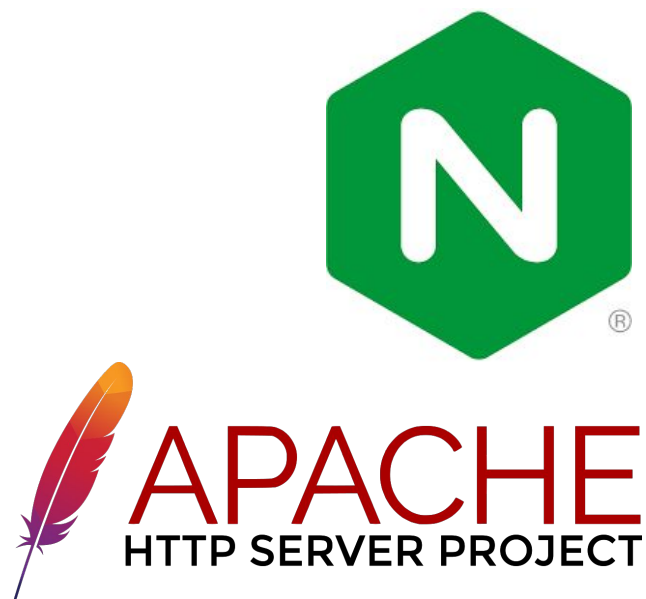
# Запуск и настройка HTTP сервера

---

# Краткий обзор серверов

Наиболее распространенными серверами сейчас являются:

- Nginx;
- Apache;
- IIS.



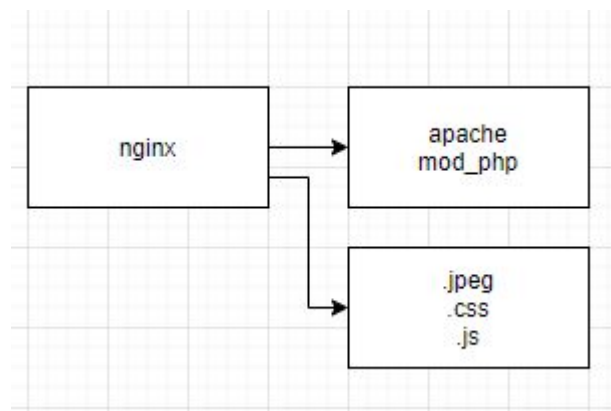
# Nginx vs. Apache

Nginx	Apache
Единый файл конфигурации.	Дополнительное конфигурирование через <a href="#">.htaccess</a> .
Обработка статики + проксирование.	Обработка динамического контента.
Модули подключаются в момент компиляции.	Динамические модули.
Асинхронная обработка запросов.	Синхронная обработка запросов.

# Nginx & Apache

→ Использование обоих инструментов позволяет добиться лучших результатов.

- Apache занимается генерацией динамического контента, в основном PHP.
- Nginx пересылает запросы на Apache и самостоятельно обрабатывает статический контент.



---

# Запуск сервера Nginx

Установка из репозитория:

```
apt-get install nginx
```

Конфигурация:

```
/etc/nginx/nginx.conf
```

Запуск | остановка | статус:

```
service nginx start | stop | status
```





# Пример конфигурации Nginx

```
user root;
worker_processes auto;
pid /run/nginx.pid;
events {
    worker_connections 1024;
}
http {
    gzip on;
    server {
        listen 80 default;
        location / {
            proxy_pass https://google.com;
        }
    }
}
```



**S for security**

---

# HTTPS

**HTTPS** (HyperText Transfer Protocol Secure) — расширение протокола HTTP, поддерживающее шифрование.

Данные, передаваемые по протоколу HTTP, «упаковываются» в криптографический протокол **SSL** или **TLS**.

В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443.



# Создание самоподписанного сертификата

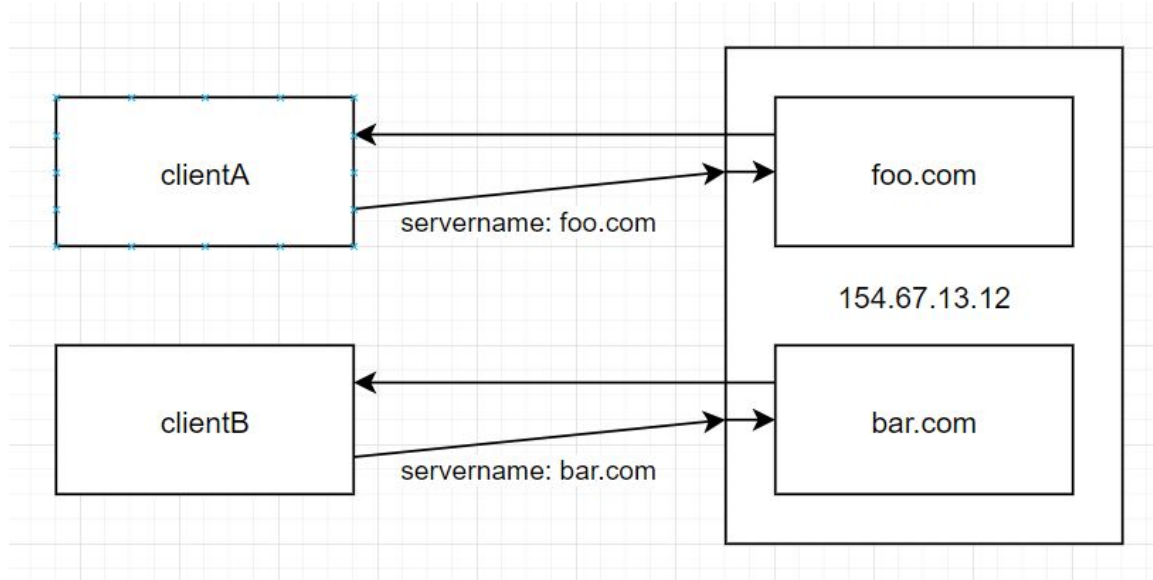
`openssl req -x509 -newkey  
rsa:4096 -keyout key.pem  
-out cert.pem -days 365` —  
сгенерировать новый  
сертификат

`openssl x509 -outform der  
-in cert.pem -out cert.crt` -  
перевести сертификат из  
pem в crt

```
user root;  
worker_processes auto;  
pid /run/nginx.pid;  
events {  
    worker_connections 1024;  
}  
http {  
    gzip on;  
    server {  
        listen 80 default;  
        listen 443 ssl;  
        server_name example.com;  
        ssl_certificate /etc/nginx/cert.crt;  
        ssl_certificate_key /etc/nginx/cert.key;  
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;  
        ssl_ciphers HIGH:!aNULL:!MD5;  
        location / {  
            proxy_pass https://google.com;  
        }  
    }  
}
```

# SNI

**Server Name Indication** (SNI) — расширение компьютерного протокола TLS, которое позволяет клиентам сообщать имя хоста, с которым он желает соединиться во время процесса «рукопожатия». Это позволяет серверу иметь несколько сертификатов на одном IP-адресе и TCP-порту и предоставлять нужный в момент рукопожатия.





# HTTP2

---

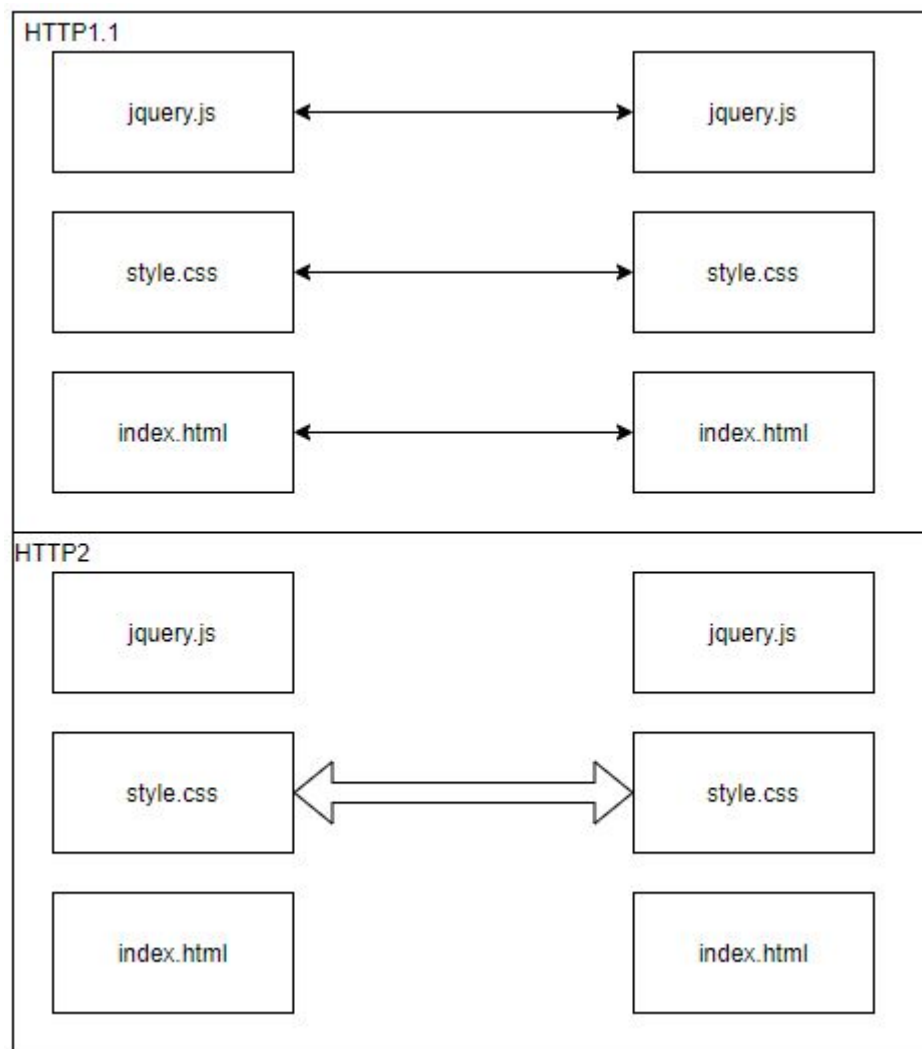
# HTTP2

## Основные цели разработки HTTP/2:

- возможность выбирать протокол, например, HTTP/1.1, HTTP/2 или другой;
- высокая совместимость с HTTP/1.1 — методы, коды статусов, поля хедеров;
- улучшение скорости загрузки благодаря сжатию хедеров запросов, бинарному протоколу, отправке данных по инициативе сервера, блокировке пакетов и запросу многократной передачи данных.

# HTTP2 Мультиплексирование

→ Использование одного TCP соединения для загрузки нескольких ресурсов с одного сервера





# HTTP2 настройка Nginx

→ Включить http2 можно только на сервере на котором настроен SSL

```
user root;
worker_processes auto;
pid /run/nginx.pid;
events {
    worker_connections 1024;
}
http {
    gzip on;
    server {
        listen 80 default;
        listen 443 ssl http2;
        server_name example.com;
        ssl_certificate /etc/nginx/cert.crt;
        ssl_certificate_key /etc/nginx/cert.key;
        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers HIGH:!aNULL:!MD5;
        location / {
            proxy_pass https://google.com;
        }
    }
}
```



**RPC**

# RPC

RPC - это удаленный вызов процедур. Стек технологий для выполнения каких либо операций на удаленном сервере.

Обычно состоит из сетевого протокола и языка описания структур.

Мы будем рассматривать реализации, основывающиеся на HTTP протоколе:

- JSON RPC;
- SOAP;
- XML RPC.

# RPC

Так как сетевой протокол у нас уже есть то различия будут лишь в разнице кодирования сообщений.

Для JSON RPC это JSON объекты:

HTTP GET {"method": "echo", "params": ["Hello JSON-RPC"], "id": 1}

200 OK {"result": "Hello JSON-RPC", "error": null, "id": 1}

---

# RPC

## XML RPC:

```
<?xml version="1.0"?>
```

```
<methodCall>
```

```
  <methodName>examples.getStateName</methodName>
```

```
  <params>
```

```
    <param>
```

```
      <value><i4>41</i4></value>
```

```
    </param>
```

```
  </params>
```

```
</methodCall>
```

# SOAP

## Пример запроса SOAP:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetails xmlns="http://warehouse.example.com/ws">
      <productID>12345</productID>
    </getProductDetails>
  </soap:Body>
</soap:Envelope>
```



# Итоги

---

# Итоги

Сегодня мы рассмотрели работу HTTP/HTTPS протокола и:

- понимаем как отправлять запросы на сервер и обрабатывать их;
- умеем настраивать веб сервера;
- понимаем способы настройки безопасного соединения.







# Домашнее задание

# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Петр Шило**