Семинар 1 **по** HTTP

Программы и утилиты: Ubuntu 20.04, telnet, curl, nginx

В этом семинаре мы научимся разными средствами делать HTTP запросы, посмотрим на различные популярные сценарии для nginx и обсудим RESTful интерфейсы.

ХОД РАБОТЫ

Структура запроса и ответа

Пусть у нас есть хороший и надёжный канал связи, то есть он гарантирует нам доставку сообщения до конечного адресата ровно один раз. На самом деле это TCP и он гарантирует нам at-least-once доставку и at-most-once обработку на стороне адресата. Мы умеем открывать соединение, надёжно слать и получать какие-то байты и закрывать соединение. Очень простая модель.

Первая версия HTTP 0.9 появилась для того, чтобы получать с какого-нибудь сервера HTML странички. Это был позапросный протокол, в котором подразумевалось открыть TCP соединение, отправить запрос в сокет, получить из него ответ и закрыть соединение. Запрос был однострочным и состоял из слова `GET` и пути на сервере:

```http
$ nc apache.org 80
GET /
```

В ответе приходил чистый HTML.

HTTP 1.0 выглядел уже более привычным для нас образом: в запросах появились заголовки, появились HEAD и POST методы, а в теле могла быть любая последовательность байт.

```http
$ nc apache.org 80
GET <url> HTTP/1.0
Header1: header 1 value
Header2: header 2 value
request data
```

Сделаем запросы руками

Попробуем сами сделать запросы разными средствами. Самый чистый способ сделать http запрос — это отправить байты в сокет, это мы сделаем с помощью `telnet`.

!**Примечание начало**:

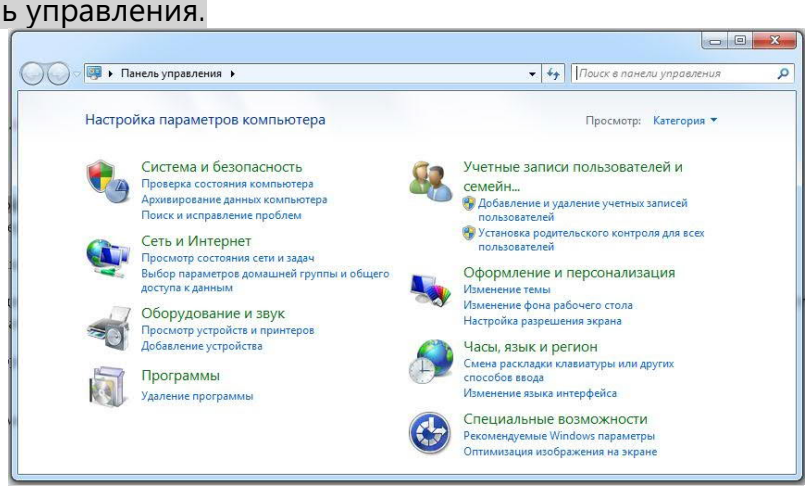
- **установить** telnet из официальных репозиториев, в Ubuntu:

sudo apt install telnet

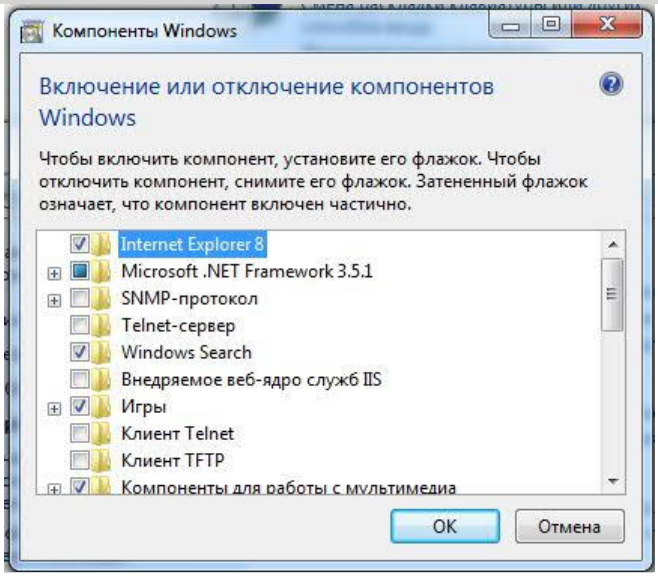
```
hdmaster@vm-ubuntu:~$ sudo apt-get install telnet
[sudo] password for hdmaster:
Reading package lists... Done
Building dependency tree
Reading state information... Done
telnet is already the newest version (0.17-41.2build1).
telnet set to manually installed.
```

Запустить TELNET на Windows 7/10 достаточно просто. Для этого необходимо сначала клиент, если он ещё не установлен:

- Зайти в Панель управления.



- Выбрать пункт «Программы».
- Выбрать вкладку «Включение или отключение компонентов Windows».



- Найти Telnet-клиент и поставить напротив него маркер, если он ещё не установлен. После нажимаем «ОК» и ждём минуту, пока клиент устанавливается.

Запуск терминала осуществляется в Windows через командную строку, если у вас не установлено каких-либо специальных утилит для работы с Телнет. Но раз вы читаете эту статью, значит, только начинаете знакомство с этой темой, и для начала неплохо бы было освоить основы управления при помощи командной строки.

1. Запускаем командную строку от имени администратора.
2. Вводим «telnet».

Командная строка перезагружается, и теперь откроется командная линия TELNET, в которой мы и будем работать.

Проверяем порт

Одно из простейших действий, выполняемых в TELNET — проверка порта. Вы можете проверить порт на наличие доступа к нему с вашего компьютера. Для этого нужно сделать следующее:

В командной строке, открытой по методу выше вводим: *telnetip-адрес номер порта*

К примеру, если ваш IP-адрес 192.168.0.1, а номер порта 21 (порт FTP), то вводим:

```
telnet 192.168.0.1 21
```

Если команда выдаёт сообщение об ошибке, значит, порт недоступен. Если появляется пустое окно или просьба ввести дополнительные данные, значит, порт открыт. Для Windows такой способ проверить порт может быть достаточно удобным.

Проверка сокета

Синтаксис выглядит следующим образом:

```
telnet localhost <номер порта>
```

Набрав в терминале telnet localhost 22, в ответ получим приветственное сообщение от SSH сервера.

Режим «отладка»

Данная опция применяется, если необходимо получить подробный отчет о каждом этапе утилиты. Сисадмин набирает в терминале:

```
sudo telnet –d localhost 22
```

Теперь во время работы утилиты с портом 22 на экране будет выводиться подробная информация о каждом действии.

Режим «Тестирование»

Другой не менее популярный вариант – тестирование веб-ресурсов. Чем удобен такой режим? Клиентская машина обрабатывает только текст, без загрузки графики. Например, введём в терминале:

telnet opennet.ru 80

Утилита отправит запрос сайту **opennet.ru** на порт под номером 80. Далее вводим ключ: GET / для показа кода страницы.

```
hdmaster@vm-ubuntu:~$ sudo telnet -d localhost 22
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.2

Invalid SSH identification string.
Connection closed by foreign host.
hdmaster@vm-ubuntu:~$ telnet opennet.ru 80
Trying 217.65.3.21...
Connected to opennet.ru.
Escape character is '^]'.
Connection closed by foreign host.
hdmaster@vm-ubuntu:~$ telnet india.colorado.edu 13
Trying 128.138.140.44...
Connected to india.colorado.edu.
Escape character is '^]'.

59373 21-06-08 17:06:15 50 0 0 610.0 UTC(NIST) *
Connection closed by foreign host.
```

!**Примечание конец**

Проверим отправку запроса Get **на стартовую страницу** mgpu.ru

```http
$ telnet mgpu.ru 80
GET / HTTP/1.1
Host: mgpu.ru
```

В ответе придёт что-то подобное:

```
hdmaster@vm-ubuntu:~$ telnet -d mgpu.ru 80
Trying 37.230.157.132...
setsockopt (SO_DEBUG): Permission denied
Connected to mgpu.ru.
Escape character is '^]'.
^CConnection closed by foreign host.
hdmaster@vm-ubuntu:~$ telnet -d mgpu.ru 23
Trying 37.230.157.132...
setsockopt (SO_DEBUG): Permission denied
^C
```

HTTP/1.0 400 Bad request

```
Cache-Control: no-cache
Connection: close
Content-Type: text/html
<html><body><h1>400 Bad request</h1>
                Your browser sent an invalid request.
                </body></html>
```

Или:

```http
HTTP/1.1 301 Moved Permanently
Server: ddos-guard
Connection: keep-alive
Keep-Alive: timeout=60
Set-Cookie: __ddg1=DSIuHDlcII8dkgOTSt0Q; Domain=.hse.ru; HttpOnly; Path=/;
Expires=Mon, 23-Aug-2021 09:41:37 GMT
Date: Sun, 23 Aug 2020 09:41:37 GMT
Content-Type: text/html
Content-Length: 162
Location: https://www. mgpu.ru/
Strict-Transport-Security: max-age=15552000
X-XSS-Protection: 1; mode=block; report=https://www. mgpu.ru.ru/n/api/xss/report
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

В ответе нам пришел 301 код и заголовок Location. Сервер попросил нас не ходить по голому http на домен mgpu.ru, а вместо этого пойти по адресу `https://www. mgpu.ru/`, иными словами открыть TCP соединение, внутри него открыть TLS соединение и после этого сделать запрос вида

```http
GET / HTTP/1.1
Host: www.mgpu.ru
```

Код 301 Moved Permanently используется как константный редирект и скорее всего в следующий раз браузер не будет делать запрос, на который был получен ответ 301. Чтобы руками не создавать TLS соединение, давайте воспользуемся утилитой `curl`.

Примечание:

Curl – это инструмент командной строки, который позволяет передавать данные с или на удаленный сервер. С помощью curl вы можете загружать или выгружать данные, используя один из поддерживаемых протоколов, включая HTTP, HTTPS, SCP, SFTP и FTP.

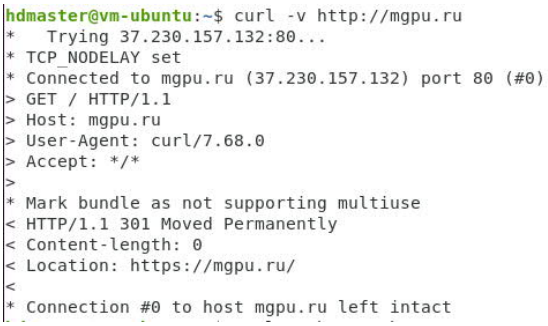
Установка Curl в Ubuntu:

sudo apt install curl

Примечание конец:

Просто `curl http://mgpu.ru` выведет в stdout тело ответа, stderr будет пустым, а мы хотим посмотреть в содержимое запроса. Для этого можно указать опцию `-v`, тогда много дополнительной информации будет выведено в stderr:

curl -v http://www.hse.ru



Проверим другой сайт:

```
hdmaster@vm-ubuntu:~$ curl -v http://hse.ru
*   Trying 186.2.163.228:80...
* TCP_NODELAY set
* Connected to hse.ru (186.2.163.228) port 80 (#0)
> GET / HTTP/1.1
> Host: hse.ru
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 301 Moved Permanently
< Server: ddos-guard
< Connection: keep-alive
< Keep-Alive: timeout=60
< Set-Cookie: __ddg1=T46TCSrKIbaEdyMtBl2C; Domain=.hse.ru; HttpOnly; Path=/; Expires=Wed, 08-Jun-2022 17:58:10 GMT
< Date: Tue, 08 Jun 2021 17:58:10 GMT
< Content-Type: text/html
< Content-Length: 162
< Location: https://www.hse.ru/
< Strict-Transport-Security: max-age=15552000
< X-XSS-Protection: 1; mode=block; report=https://www.hse.ru/n/api/xss/report
<
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host hse.ru left intact
```

Видим, что нас опять, как и в предыдущий раз, просят проследовать по новому урлу.

curl -v https://www.hse.ru

```
hdmaster@vm-ubuntu:~$ curl -v https://www.hse.ru
*   Trying 186.2.163.228:443...
* TCP_NODELAY set
* Connected to www.hse.ru (186.2.163.228) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
  CApath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_128_GCM_SHA256
* ALPN, server accepted to use h2
* Server certificate:
*  subject: CN=*.hse.ru
*  start date: Dec 26 00:00:00 2019 GMT
*  expire date: Jan 29 23:59:59 2022 GMT
*  subjectAltName: host "www.hse.ru" matched cert's "*.hse.ru"
*  issuer: C=GB; ST=Greater Manchester; L=Salford; O=Sectigo Limited; CN=Sectigo RSA Domain Validation Secure Serve
*  SSL certificate verify ok.
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* Using Stream ID: 1 (easy handle 0x55b3a541ae10)
> GET / HTTP/2
> Host: www.hse.ru
> user-agent: curl/7.68.0
> accept: */*
>
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
* Connection state changed (MAX_CONCURRENT_STREAMS == 128)!
< HTTP/2 302
< server: ddos-guard
< set-cookie: __ddg1=FIFNMbXoUMSM9hAAUtkB; Domain=.hse.ru; HttpOnly; Path=/; Expires=Wed, 08-Jun-2022 18:05:57 GMT
< date: Tue, 08 Jun 2021 18:05:57 GMT
< content-type: text/html
< content-length: 138
< location: https://www.hse.ru/en/
< expires: Tue, 08 Jun 2021 18:05:57 GMT
< cache-control: max-age=0
< strict-transport-security: max-age=15552000
< x-xss-protection: 1; mode=block; report=https://www.hse.ru/n/api/xss/report
< set-cookie: tracking=ZEsUBGC/sYWxvSWKCMDwAg==; expires=Thu, 31-Dec-37 23:55:55 GMT; domain=.hse.ru; path=/
<
<html>
<head><title>302 Found</title></head>
<body>
<center><h1>302 Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host www.hse.ru left intact
```

Тут мы видим 302 в ответе, это похоже на 301, но 302 говорит о том, что для данного запроса был найден новый путь, куда надо проследовать и возможно повторный запрос даст 302 на другую страницу (такое бывает). В ответе видно, что сервер решил, что мы англоязычный клиент и хотим читать английскую версию сайта вышки. Ну действительно, давайте сделаем запрос туда и получим свой долгожданный 200.

```
hdmaster@vm-ubuntu:~$ curl -v https://www.hse.ru/en/
*   Trying 186.2.163.228:443...
* TCP_NODELAY set
* Connected to www.hse.ru (186.2.163.228) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
  CApath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_128_GCM_SHA256
* ALPN, server accepted to use h2
* Server certificate:
*  subject: CN=*.hse.ru
*  start date: Dec 26 00:00:00 2019 GMT
*  expire date: Jan 29 23:59:59 2022 GMT
*  subjectAltName: host "www.hse.ru" matched cert's "*.hse.ru"
*  issuer: C=GB; ST=Greater Manchester; L=Salford; O=Sectigo Limited;
*  SSL certificate verify ok.
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgr
* Using Stream ID: 1 (easy handle 0x55f47b0a4e10)
> GET /en/ HTTP/2
> Host: www.hse.ru
> user-agent: curl/7.68.0
> accept: */*
>
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
* Connection state changed (MAX_CONCURRENT_STREAMS == 128)!
< HTTP/2 200
< server: ddos-guard
< set-cookie: __ddg1=YmchtgbtE1xnk21SNASx; Domain=.hse.ru; HttpOnly; P
```

Прекрасно, мы получили ответ 200, причём curl выбрал HTTP/2 для запроса и мы видим новую версию в тексте.

Теперь давайте повторим этот же **запрос через браузер** и пронаблюдаем воочию **В ПАПКЕ С ПРАКТИЧЕСКИМ ЗАДАНИЕМ НАХОДИТСЯ** media

Видимо, судя по отправленному IP в заголовке сервер понял, что мы из России и сразу показал нам русскую версию сайта, чего не произошло, когда мы делали сырой curl. Тем не менее, после `http://hse.ru` нас отправили на

`https://www.hse.ru/`, браузер отработал этот редирект и начал получать нормальную страницу и данные на ней.

HTTP **серверы**

Имеет смысл разделить HTTP серверы на два вида:
1. Для раздачи статических файлов (html, css, js, медиа), проксирования запросов и полной поддержки протокола.
   Пример: Apache, nginx, Traefik
2. Кастомные серверы, реализующие произвольное поведение ответа на запросы с помощью какой-нибудь библиотеки.

   Популярные библиотеки: flask (python), aiohttp (python), phantom (c), spring (java)

Далее мы настроим простейшие сценарии в nginx и посмотрим в примеры RESTful API, которые реализуют кастомные серверы.

Nginx

Почему-то среди начинающих разработчиков есть ощущение, что nginx это что-то сложное. На самом деле конфигурации для простейших сценариев занимают меньше 10 строк и предельно понятны.

Сначала поставим `nginx` на вашу операционную систему.

Ubuntu/Debian:
```
$ sudo apt update && sudo apt install -y nginx
```

OS X:
```
$ brew install nginx
```

На OS X настраивать nginx не так приятно, как на Linux, поэтому примеры ниже будут валидны для Linux.

Вообще, если у вас появилось желание поставить сырой nginx на OS X, то что-то идёт не так. Для локального тестирования лучше использовать docker контейнер с nginx, в него легко подсовывать статику или кастомные конфигурации (в т.ч. с проксированием в соседний контейнер). Подробнее можно прочитать в Docker Hub https://hub.docker.com//nginx._

После установки должна появиться папка `/etc/nginx`, в которой мы и будем создавать конфигурации. В `/etc/nginx` есть папки `sites-available` и `sites-enabled`. Это одни и те же конфигурации, только в `sites-available` находятся все доступные пользовательские конфигурации, а в `sites-enabled` добавляются ссылки на

конфигурации, которые надо включить в данный момент у сервера. Там уже лежит конфигурация `default` и она включена:

```
$ ls -l /etc/nginx/sites-enabled/
total 0
lrwxrwxrwx 1 root root 34 May 31 15:33 default -> /etc/nginx/sites-available/default
$ ls -l /etc/nginx/sites-available/
total 4
-rw-r--r-- 1 root root 2072 May 31 15:37 default
```

Таким образом, мы будем писать конфигурации в `sites-available`, а потом добавлять ссылки на них в `sites-enabled`.

Чтобы не конфликтовать с `default`, давайте сразу удалим его из `sites-enabled`:

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

```
hdmaster@vm-ubuntu:~$ ls -l /etc/nginx/sites-enabled/
total 0
lrwxrwxrwx 1 root root 34 июн  8 22:16 default -> /etc/nginx/sites-available/default
hdmaster@vm-ubuntu:~$ ls -l /etc/nginx/sites-available/
total 4
-rw-r--r-- 1 root root 2416 мар 26  2020 default
hdmaster@vm-ubuntu:~$ sudo rm /etc/nginx/sites-enabled/default
[sudo] password for hdmaster:
hdmaster@vm-ubuntu:~$ ls -l /etc/nginx/sites-available/
total 4
-rw-r--r-- 1 root root 2416 мар 26  2020 default
hdmaster@vm-ubuntu:~$ ls -a /etc/nginx/sites-available/default
/etc/nginx/sites-available/default
```

Раздача статики

Самая простая задача, которую можно реализовать с помощью nginx — захостить статические файлы. Будет странно это использовать, чтобы поделиться с кем-то файлом по сети, но, например, захостить простейший личный сайт или скомпилированное JS приложение можно именно так.

Для начала создадим простую статику, которую можно будет раздать — html файл и картинку. Это принято делать в `/var/www/your-website.com`:

```
$ sudo mkdir -p /var/www/simple_static
$ sudo chown hdmaster /var/www/simple_static
$ cd /var/www/simple_static
```

```
$ curl https://www.google.com/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png > google.png
$ printf "<body>This is our first html file</body>\n" > index.html
```

Теперь в папке `/var/www/simple_static` есть два файла:

```
$ ls
google.png  index.html
```

Теперь напишем конфигурацию nginx, которая позволит получить доступ к этим файлам по http. Создадим файл `/etc/nginx/sites-available/simple_static.conf` и напишем в него очень простую конфигурацию:

$sudo service nginx stop

Или $sudo systemctl stop nginx

```nginx
server {
  listen 80 default_server;
  server_name _;
  root /var/www/simple_static;
}
```

Теперь положим ссылку на файл в `sites-enabled` и перезагрузим nginx:

```
$ cd /etc/nginx/
$ sudo ln sites-available/simple_static.conf sites-enabled/simple_static.conf
$ sudo nginx -s reload
$sudo systemctl stop nginx
```

hdmaster@vm-ubuntu:~$ sudo service nginx stop

[sudo] password for hdmaster:

hdmaster@vm-ubuntu:~$ cd /var/www/simple_static

hdmaster@vm-ubuntu:/var/www/simple_static$ ls

google.png index.html

hdmaster@vm-ubuntu:/var/www/simple_static$ cd

hdmaster@vm-ubuntu:/$ sudo nano /etc/nginx/sites-available/simple_static.conf

```
server {
  listen 80 default_server;
  server_name _;
  root /var/www/simple_static;
}
```

[sudo] password for hdmaster:

hdmaster@vm-ubuntu:/$ cd /etc/nginx/

hdmaster@vm-ubuntu:/etc/nginx$ sudo ln sites-available/simple_static.conf sites-enabled/simple_static.conf

hdmaster@vm-ubuntu:/etc/nginx$ sudo nginx -t

nginx: [emerg] unknown directive "nginx" in /etc/nginx/sites-enabled/simple_static.conf:3

nginx: configuration file /etc/nginx/nginx.conf test failed

hdmaster@vm-ubuntu:/etc/nginx$ cd

hdmaster@vm-ubuntu:/$ sudo nano /etc/nginx/sites-available/simple_static.conf

hdmaster@vm-ubuntu:/$ cd /etc/nginx/

hdmaster@vm-ubuntu:/etc/nginx$ sudo nginx -t

nginx: the configuration file /etc/nginx/nginx.conf syntax is ok

nginx: configuration file /etc/nginx/nginx.conf test is successful

hdmaster@vm-ubuntu:/etc/nginx$ sudo service nginx start

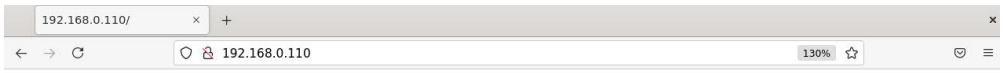
Сделаем запрос:

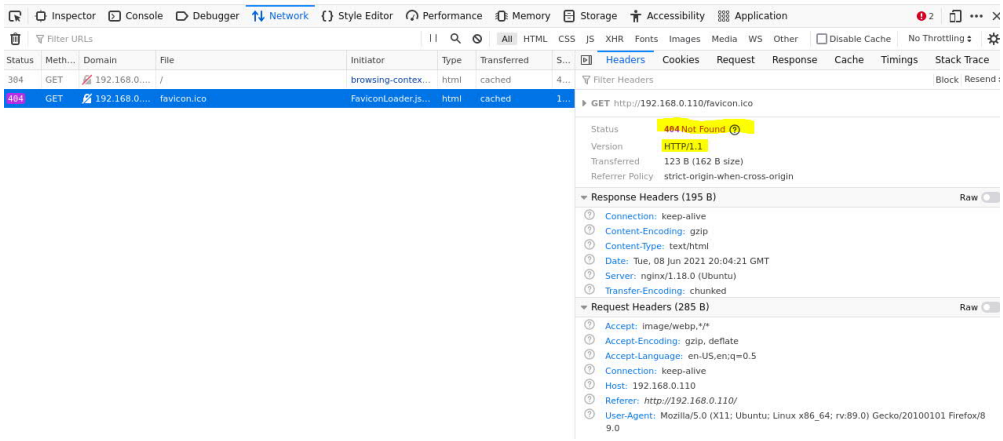
<details>

  <summary><code>$ curl -v http://localhost:80/</code></summary>

```
hdmaster@vm-ubuntu:/etc/nginx$ cd
hdmaster@vm-ubuntu:~$ curl -v http://localhost:80/
*   Trying 127.0.0.1:80...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET / HTTP/1.1
> Host: localhost
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.18.0 (Ubuntu)
< Date: Tue, 08 Jun 2021 20:40:25 GMT
< Content-Type: text/html
< Content-Length: 41
< Last-Modified: Tue, 08 Jun 2021 19:41:46 GMT
< Connection: keep-alive
< ETag: "60bfc7fa-29"
< Accept-Ranges: bytes
<
<body>This is our first html file</body>
* Connection #0 to host localhost left intact
hdmaster@vm-ubuntu:~$
```

Через браузер смотрим:





Стоит отметить, что несмотря на то, что мы не указали index.html в запросе, он всё равно подсунулся. Это происходит, так как по умолчанию в nginx открывается страница `index.html`, если она есть в корне каталога `root`.

Сделаем запрос за картинкой и спрячем содержимое картинки в `/dev/null`:

<details>
 <summary><code>$ curl -v http://localhost:80/google.png > /dev/null</code></summary>

```
hdmaster@vm-ubuntu:~$ curl -v http://localhost:80/google.png > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*   Trying 127.0.0.1:80...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /google.png HTTP/1.1
> Host: localhost
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.18.0 (Ubuntu)
< Date: Tue, 08 Jun 2021 20:52:08 GMT
< Content-Type: image/png
< Content-Length: 13504
< Last-Modified: Tue, 08 Jun 2021 19:41:26 GMT
< Connection: keep-alive
< ETag: "60bfc7e6-34c0"
< Accept-Ranges: bytes
<
{ [13504 bytes data]
100 13504  100 13504    0     0  1465k      0 --:--:-- --:--:-- --:--:-- 1465k
* Connection #0 to host localhost left intact
```

Множество остальных примеров конфигураций и любые запросы можно задавать в google — nginx это самый популярный на сегодняшний день веб-сервер, поэтому инструкций великое множество.

HTTP **проксирование(самостоятельно)**

Очень важной и удобной возможностью nginx является http проксирование. Если у вас есть небольшой проект, доступный по http, то вместо прямого доступа скорее всего вы хотите спрятать его за nginx.

Пусть у нас локально на закрытом для внешнего мира порте работает какое-нибудь приложение. Мы хотим, чтобы nginx принимал запросы на конкретный домен `amazing-domain.com` по 80 порту (http) и перенапрвлял их в наше приложение. Для этого нужно все запросы от корня направить в наше приложение:

```nginx
server {
  listen 80;
  server_name amazing-domain.com;
  location /  {
    proxy_pass http://localhost:8123/
  }
}
```

```

Ещё понятнее, зачем нужно проксирование, на следующем примере. Допустим, у вас есть два сервиса — один раздаёт статику (html файлы, стили, скрипты, картинки, медиа), другой отвечает на различные запросы. Чтобы они были спрятаны за один домен, но на разных путях, то есть чтобы `amazing-domain.com` раздавал статику, а `amazing-domain.com/api/` вёл в бэкенд, можно взять первый сценарий с раздачей статики и добавить к нему `location /api/` с `proxy_pass` на любой урл для бэкенда, даже на другом сервере.

---

На самом деле, сценариев и настроек для использования nginx великое множество. Можно настраивать локальный SSL сертификат через lets-encrypt утилиту, можно крутить настройки запросов, заголовков и т.д. Рекомендую любую мысль "хочу такую настройку" загуглить, скорее всего вы найдете решение.

### **Быстрая обработка клиентских запросов**

Мы тут обсуждаем HTTP, запросы, серверы, которые обрабатывают запросы. Однако, вам так или иначе предстоит столкнуться с большой нагрузкой и в связи с этим хотелось бы разобрать, какие трудности при различных подходах обработки возникают, а так же какие существуют подходы к обработке большого числа запросов. На самом деле, на семинаре мы это расскажем, а в текстовой версии даём ссылку на хороший разбор на русском языке, так как это много раз уже рассказано: https://iximiuz.com/ru/posts/writing-python-web-server-part-2/

### REST & RESTful API

_REST_ **или** _Representational State Transfer_ — аббревиатура, которую знает любой разработчик веб-серверов. Формально говоря, это архитектурный подход для построения модели взаимодействия клиента и сервера. Звучит странно, но на самом деле всё просто — существует ряд принципов, следуя которым мы получим формально _REST_ приложение, вам скорее всего рассказывали о них на лекции, но их можно найти даже на

[википедии](https://en.wikipedia.org/wiki/Representational_state_transfer). На деле же в индустрии сформировалась не просто лучшая, а скорее даже единственная устоявшаяся практика, согласно которой REST приложения реализуются с помощью HTTP, передают данные в форме JSON или XML, а так же следуют ряду принципов построения RESTful API.
```

После того, как мы научились раздавать статику через `GET` запросы с nginx сервера, можно догадаться, что похожим образом могут быть организованы модифицирующие операции. Например, добавить или удалить файл. Действительно, HTTP поддерживает так же другие методы: `POST`, `HEAD`, `PUT`, `PATCH`, `DELETE`. В HTTP такой подход работы с данными называется WebDAV, он позволяет иметь полноценный доступ к удалённым файлам — читать, модифицировать и удалять. В nginx тоже можно как-то включить это, но сейчас не об этом. Помимо файлов существуют другие объекты, которыми мы бы хотели управлять.

На самом деле этот разговор начинает напоминать CRUD, но RESTful это не всегда CRUD. CRUD это акроним для Create Read Update Delete. Это один из паттернов дизайна RESTful API, например:

- `GET /articles` – Список доступных статей, возможно с пагинацией и фильтрацией, настраиваемыми через GET-параметры запроса, например `GET /articles?limit=10&offset=5&author=Albert`;

- `POST /articles` – Создает статью из тела запроса;

- `GET /articles/{id}` – Статья с идентификатором `id`;

- `PUT /articles/{id}` - Полностью обновить существующую статью `id`;

- `PATCH /articles/{id}` – Частично обновить статью `id`;

- `DELETE /articles/{id}` – Удалить статью `id`.

Стоит отметить, что CRUD паттерн действительно удовлетворяет формальным требованиям REST: сервер не хранит состояние пользователя, данные могут кэшироваться на уровне HTTP, сервер умеет отвечать большому количеству клиентов, клиенты общаются с сервером одним и тем же форматом. Но можно придумать множество других модификаций формата или совсем отходящих от него веток, например, у Facebook API весьма похожий, но не совсем такой интерфейс: [документация](https://developers.facebook.com/docs/graph-api/reference/v2.2/user). Там Graph API, где к вершинам графа можно обращаться с помощью CRUD. Очень интересный и хорошо спроектированный пример RESTful API.

Или можно посмотреть на совсем странный пример: движок рекламы Яндекса. Допустим мы хотим получить рекламу для поиска по запросу «окна», для этого делается `GET` запрос в ручку `code`:

```
```

```
GET /code/2?text=окна HTTP/1.1
Host: yabs.yandex.ru
```

В ответе в теле баннеров будут ссылки, которые надо вызвать в случае, если пользователь кликает на тело баннера. Например

```
http://yabs.yandex.ru/count/WSuejI_zO6q19Gu051Stei35Lkz4TWK0RG8GWhY04343bbPV000003W4G0H81hgcou3h5O01-hiDY07cmiUFJf01XCwA_JAO0V2En-Grk06Wu8lp6y01PDW1qC236E01XhRx5kW1hW6W0hRBfnVm0i7Krk8Bc0FOp1gm0mJe1AI_0lW4c5o81PXSa0NJcG6W1P0Sg0Mu5x05k1Uu1OWdm0NJcG781OWdiI_DiQa7vNU46I_ctqYm1u20c0ou1u05yGUqEtIjdS1vmeI2s-N92geB4E20U_IbTm00AYM2yhwk1G3P2-WBc5pm2mM83D2Mthu1gGm00000miPbI-WC0U0DWu20GA0Em8GzeG_Pu0y1W12oXFiJ2lWG4e0H3GuRQ4gjsDi-y18Ku1E89w0KY2Ue5DEPlA74y0Ne50pG5RoXnF05s1N1YlRieu-y_6Fme1RGZy3w1SaMq1RGbjw-0O4Nc1VhdlmPm1SKs1V0X3sP6A0O1h0Oe-hP-WNG604O07QIHKDExP6popQOGFM8ydnMIQvapWs9jSugkDC8US5dVMyI9XpusjyR4_mgs44iCM2i6DqnsBZ9P0JvEcVcbgGBUI9ocBBODOmZeiW3V080~1?from=&q=%D0%BE%D0%BA%D0%BD%D0%B0&etext=
```

По ней можно кликнуть и где-то на серверах рекламы будет записан лог о том, что вы как пользователь кликнули по заголовку. Совершенно непонятно, что происходит. И на самом деле, простому пользователю и не должно быть понятно. Оно далеко от CRUD, хотя бы потому, что оно почти Read-Only, однако это тоже можно назвать RESTful API, потому что оно соблюдает основные принципы.