МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

Урок №2

Протокол HTTP, веб-сервер

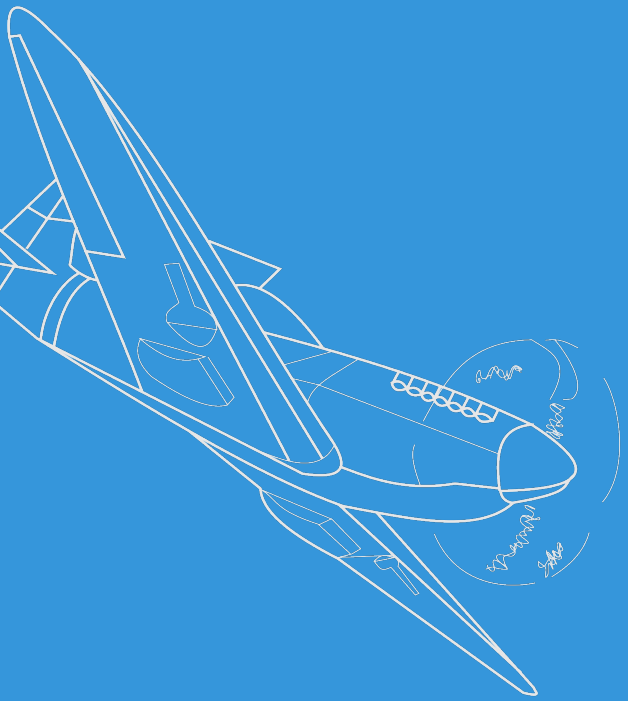
на котором расскажут про что такое URL, HTTP, чем HTTP отличается от HTTPS, предназначение и особенности веб-сервера, как настроить nginx.

21 сентября 2020 года

Антон Кухтичев

Содержание занятия

1. Поговорим, что такое URL;
2. Подробнее обсудим протокол HTTP;
3. Расскажут, что такое веб-сервер;
4. Конфигурационный файл nginx;
5. Сервер приложений;
6. WSGI-протокол.

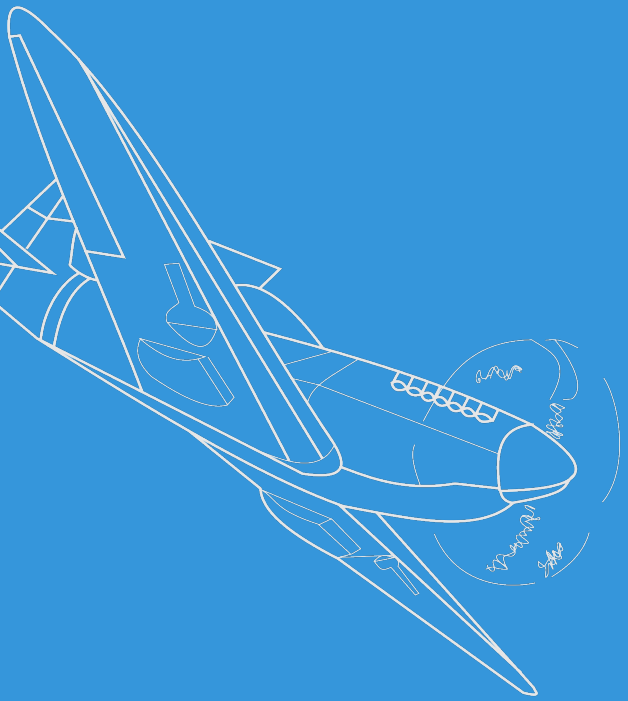


Документы

Документы

Документы могут быть:

- Статические
  - Это файлы на дисках сервера;
  - Как правило, обладают постоянным адресом.
- Динамические
  - Создаются на каждый запрос;
  - Содержимое зависит от времени и пользователя;
  - Адрес может быть постоянным или меняться.



URI, URL, URN

Определения

- **URI** – Uniform Resource Identifier (унифицированный идентификатор ресурса);
- **URL** – Uniform Resource Locator (унифицированный локатор/указатель ресурса);
- **URN** – Uniform Resource Name (унифицированное имя ресурса).

URI является либо URL, либо URN, либо одновременно обоими.

URN — uniform resource name

`urn:<NID>:<NSS>`

- `<NID>` — идентификатор пространства имён;
- `<NSS>` — строка из определённого пространства имён.

Пример:

- `urn:isbn:5170224575` — URN книги, идентифицируемой номером ISBN;

URL — uniform resource locator

`<схема>://[//[<логин>[:<пароль>]@]<хост>[:<порт>]][/<URL-путь>][?<параметры>][#<якорь>]`

`http://server.org:8080/path/doc.html?a=1&b=2#part1`

- `http` — протокол;
- `server.org` — DNS имя сервера (может указываться ip-адрес машины);
- `8080` — TCP порт;
- `/path/doc.html` — путь к файлу;
- `a=1&b=2` — параметры запроса;
- `part1` — якорь, положение на странице.

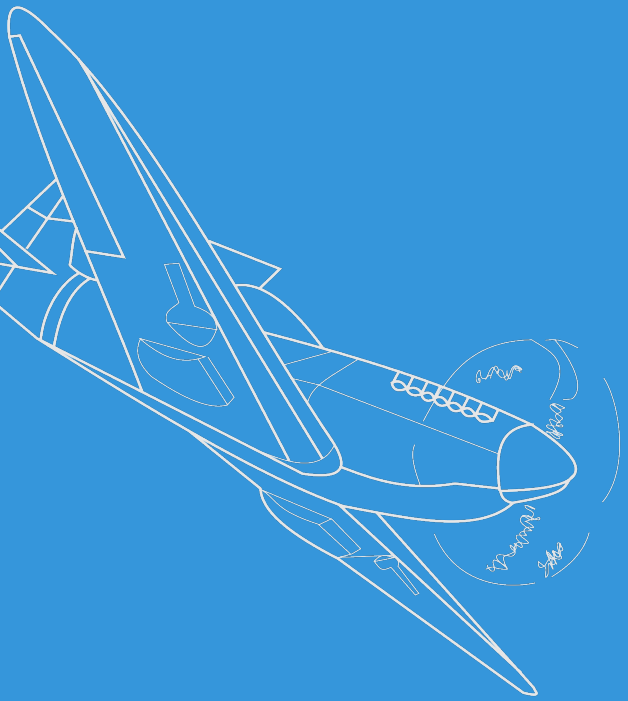
Абсолютные и относительные URL

- `http://server.org/1.html` — абсолютный;
- `//server.org/1.html` — абсолютный (schemeless);
- `/another/page.html?a=1` — относительный (в пределах домена);
- `pictures/cat.png` — относительный (от URL текущего документа);
- `?a=1&b=2` — относительный (от URL текущего документа);
- `#part2` — относительный (в пределах текущего документа);

Правила разрешения URL

`https://site.com/path/page.html` — основной документ

- `http://wikipedia.org` = `http://wikipedia.org`
- `//cdn.org/jquery.js` = `https://cdn.org/jquery.js`
- `/admin/index.html` = `https://site.com/admin/index.html`
- `another.html` = `https://site.com/path/another.html`
- `?full=1` = `https://site.com/path/page.html?full=1`
- `#chapter2` = `https://site.com/path/page.html#chaprer2`



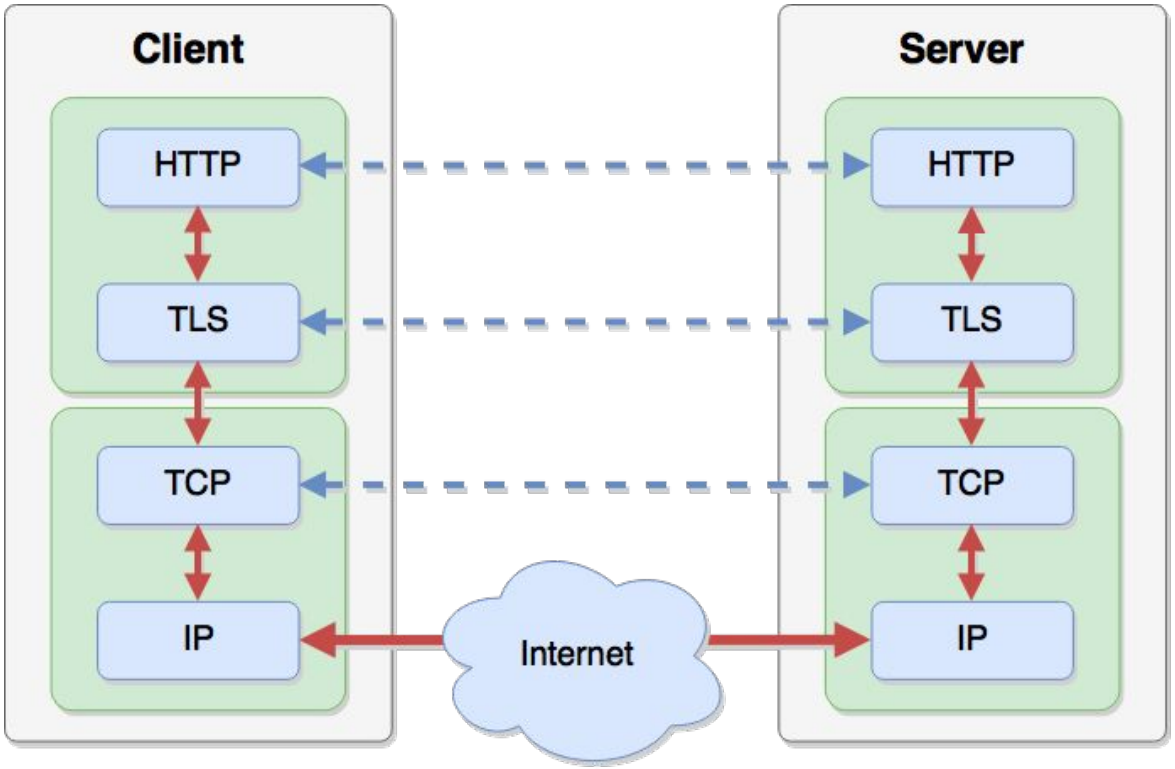
HTTP-протокол

Особенности, методы, версии, заголовки.

Как задачи решает HTTP?

- Передача документов;
- Передача мета-информации;
- Авторизация;
- Поддержка сессий;
- Кеширование документов;
- Согласование содержимого (negotiation);
- Управление соединением.

Как происходит HTTP запрос?



Ключевые особенности HTTP

- Работает поверх TCP/TLS;
- Протокол запрос-ответ;
- Не поддерживает состояние (соединение) — stateless;
- Текстовый протокол;
- Расширяемый протокол.

HTTP запрос состоит из

- строка запроса:
  - метод,
  - URL документа,
  - версия.
- заголовки;
- тело запроса;

HTTP/1.0 запрос


```
GET http://www.ru/robots.txt HTTP/1.0
Accept: text/html, text/plain
User-Agent: telnet/hands
If-Modified-Since: Fri, 24 Jul 2015 22:53:05 GMT
```

Перевод строки — `\r\n`

HTTP/1.1 запрос


```
GET /robots.txt HTTP/1.1
Accept: text/html,application/xhtml+xml
Accept-Encoding: gzip, deflate
Cache-Control: max-age=0
Connection: keep-alive
Host: www.ru
User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/39.0
```

HTTP/1.1 ответ


```
HTTP/1.1 404 Not Found
Server: nginx/1.5.7
Date: Sat, 25 Jul 2015 09:58:17 GMT
Content-Type: text/html; charset=iso-8859-1
Connection: close

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>...
```

Методы HTTP-запроса

- `GET` — получение документа;
- `HEAD` — получение только заголовков;
- `POST` — отправка данных на сервер;
- `PUT` — отправка документа на сервер;
- `DELETE` — удаление документа;
- `CONNECT`, `TRACE`, `OPTIONS` — используются редко;
- `COPY`, `MOVE`, `MKCOL` — расширения WebDAV.

HTTP-коды ответов

- `1xx` — информационные;
- `2xx` — успешное выполнение;
- `3xx` — перенаправления;
- `4xx` — ошибка на стороне клиента;
- `5xx` — ошибка на стороне сервера.

HTTP-коды ответов (1)

`200 OK` – запрос успешно выполнен;

`204 No Content` – запрос успешно выполнен, но документ пуст;

`301 Moved Permanently` – документ сменил URL;

`302 Found` – повторить запрос по другому URL;

`304 Not Modified` – документ не изменился, использовать кеш.

HTTP-коды ответов (2)

`400 Bad Request` — неправильный синтаксис запроса;

`401 Unauthorized` — требуется авторизация;

`403 Forbidden Moved Permanently` — нет доступа (неверная авторизация);

`404 Not Found` — документ не найден;

`500 Internal Server Error` — неожиданная ошибка сервера;

`502 Bad Gateway` — проксируемый отвечает с ошибкой;

`504 Gateway Timeout` — проксируемый сервер не отвечает;

Заголовки HTTP (общие)

Для управления соединением и форматом сообщения (документа):

- `Content-Type` — Mime тип документа;
- `Content-Length` — длина сообщения;
- `Content-Encoding` — кодирование документа, например, gzip-сжатие;
- `Transfer-Encoding` — формат передачи, например, chunked;
- `Connection` — управление соединением;
- `Upgrade` — смена протокола.

Заголовки HTTP-запросов

- `Authorization` — авторизация, чаще всего логин/пароль;
- `Cookie` — передача состояния (сессии) на сервер;
- `Referer` — URL предыдущего документа, контекст запроса;
- `User-Agent` — описание web-клиента, версия браузера;
- `If-Modified-Since` — условный GET запрос;
- `Accept-*` — согласование (negotiation) содержимого.

Заголовки HTTP-ответов

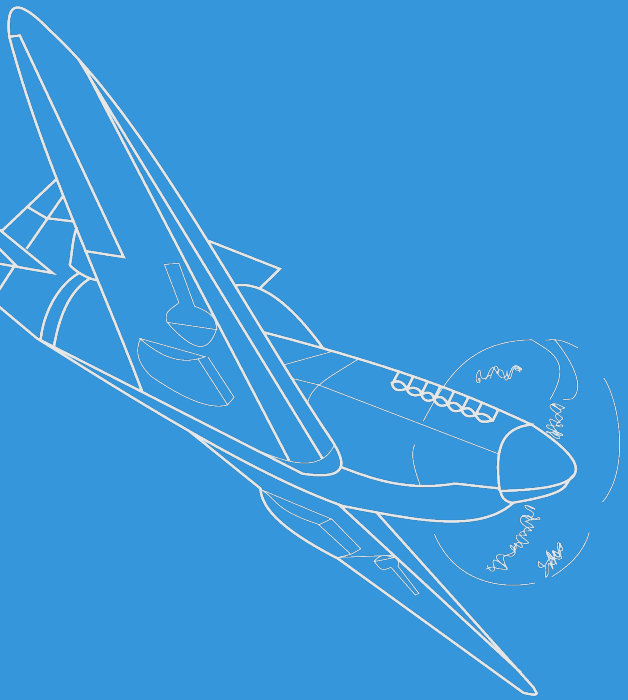
- `Location` — новый URL документа при перенаправлениях (коды 301, 302);
- `Set-Cookie` — установка состояния (сессии) в браузере;
- `Last-Modified` — дата последнего изменения документа;
- `Date` — Дата на сервере, для согласования кешей;
- `Server` — описание web-сервера, название и версия.

Логика управления в HTTP/1.1

Соединение должно быть закрыто, если:

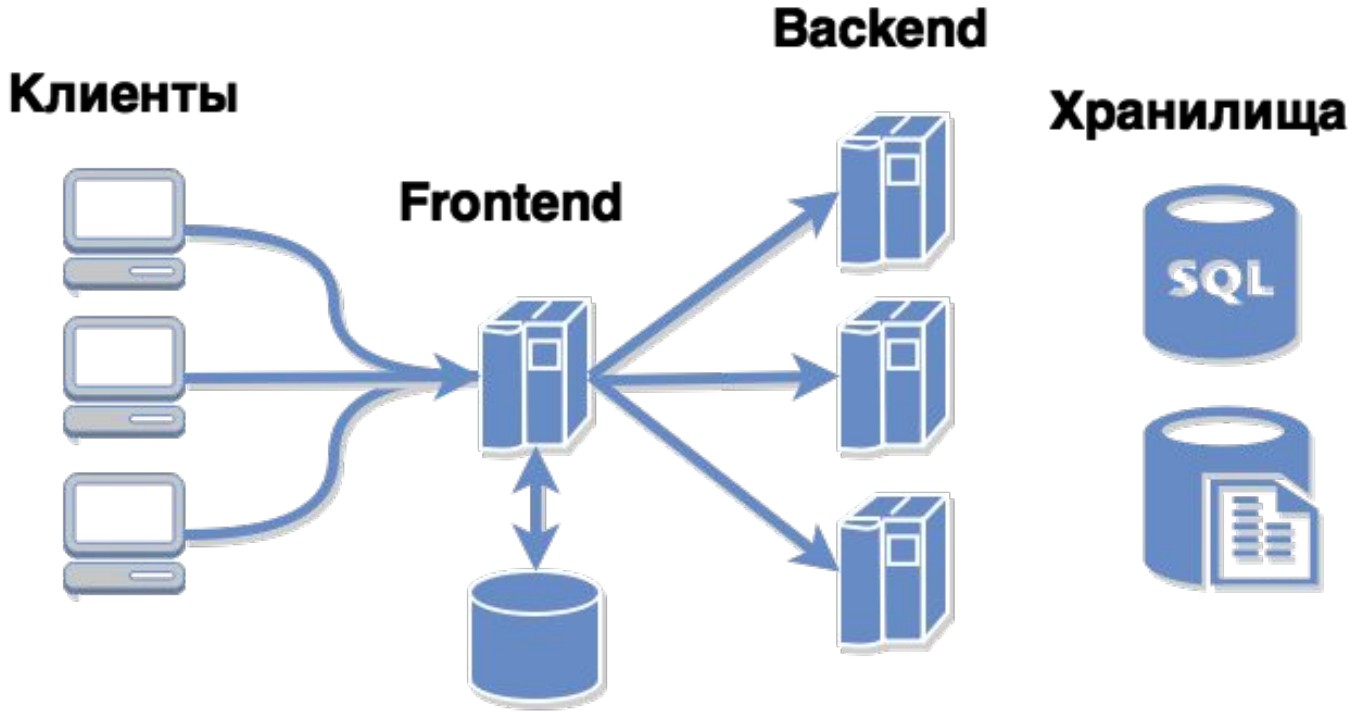
- сервер или клиент использует HTTP младше 1.1;
- сервер или клиент передал заголовок Connection: close;
- по истечении таймаута (обычно небольшой, около 10 с);

Иначе соединение остается открытым для последующих запросов.



Трёхзвенная архитектура

Общая архитектура



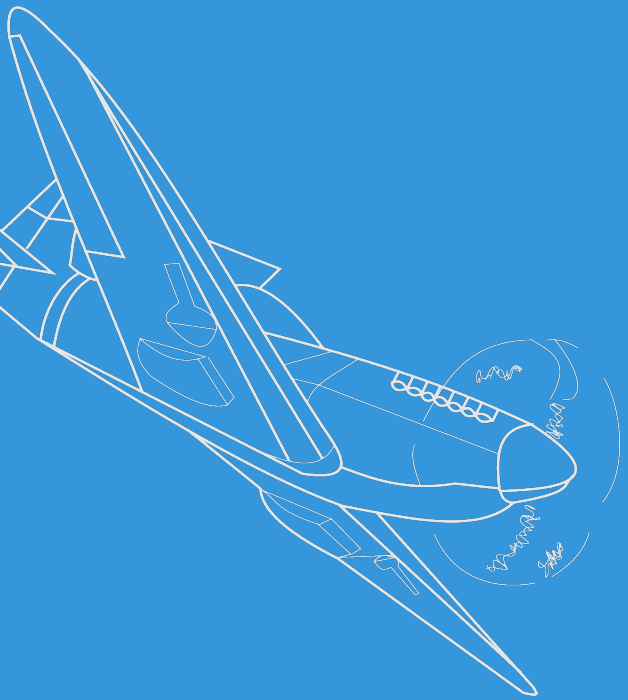
Задача Frontend (web) сервера

- отдача статических документов;
- проксирование (reverse proxy);
- балансировка нагрузки;
- кеширование;
- сборка SSI;
- авторизация, SSL, нарезка картинок, gzip.

Reverse proxy

- frontend (медленно) читает запрос от клиента;
- frontend (быстро) передает запрос свободному backend;
- backend генерирует страницу;
- backend (быстро) возвращает ответ frontend серверу;
- frontend (медленно) возвращает ответ клиенту.

Результат: backend занят минимально возможное время.



Веб-сервер

Веб-сервера


NGINX
http server


APACHE
HTTP SERVER


LIGHTTPD
fly light.


Microsoft
IIS

Запуск веб-сервера

```
# Установка в Ubuntu
sudo apt install nginx
# Установка в MacOS
brew install nginx
```

- Команда на запуск;

  `sudo /etc/init.d/nginx start`
- Чтение файла конфигураций;
- Получение порта 80;
- Открытие (создание) логов;
- Понижение привилегий;
- Запуск дочерних процессов/потоков;
- Готов к обработке запросов.

Файлы веб-сервера

Конфиг `/etc/nginx/nginx.conf` (для Ubuntu)

Конфиг `/usr/local/etc/nginx/nginx.conf` (для MacOS)

Init-скрипт `/etc/init.d/nginx [start|stop|restart]`

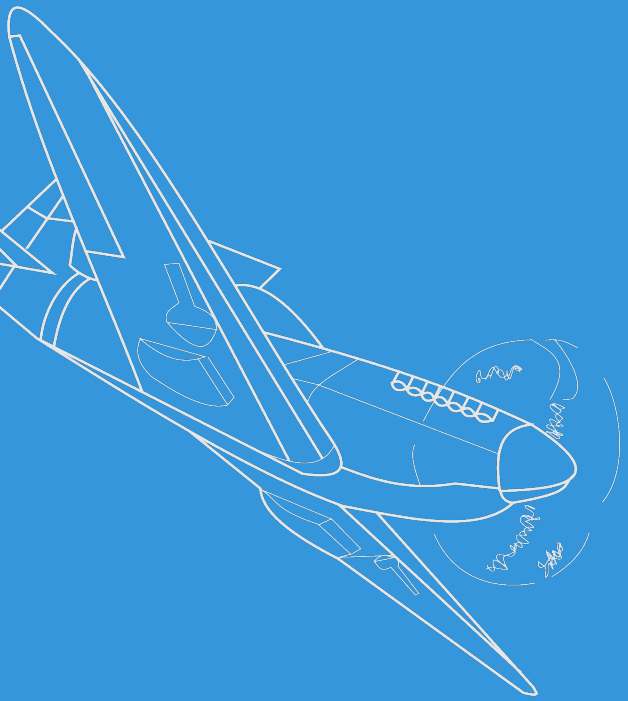
PID-файл `/var/run/nginx.pid`

Error-лог `/var/log/nginx/error.log`

Access-лог `/var/log/nginx/access.log`

Процессы веб-сервера

- Master (root, 1 процесс)
  - Чтение и валидация конфига;
  - Открытие сокета(ов) и логов;
  - Запуск и управление дочерними процессами (worker);
  - Graceful restart, Binary updates.
- Worker (nobody, 1+процессов)
  - Обработка входящих запросов.



Конфигурация веб-сервера

Терминология

`virtual host, вирт. хост` – секция конфига web сервера, отвечающая за обслуживание определенного домена.

`location` – секция конфига, отвечающая за обслуживание определенной группы URL.

Структура конфига nginx

- nginx состоит из модулей, которые настраиваются директивами;
- директивы:
  - простые (worker_processes 2;)
  - блочные (http{ server{} })
- `http`, `events` внутри `main`, `server` внутри `http`, `location` внутри `server`.

Основные директивы

- `user` – пользователь, от лица которого будут запущены;
- `worker_processes` – количество дочерних процессов;
- `error_log` – файл, в который записываются ошибки и уровень ошибок;
- `http` – конфиг веб-сервера;
- `include` – включает содержимое файла;
- `log_format` – формат записи в access_log;
- `server` – virtual host;

Приоритеты location

1. `location = /img/1.jpg`
2. `location ^~ /pic/`
3. `location ~* \.jpg$`
4. `location /img/`

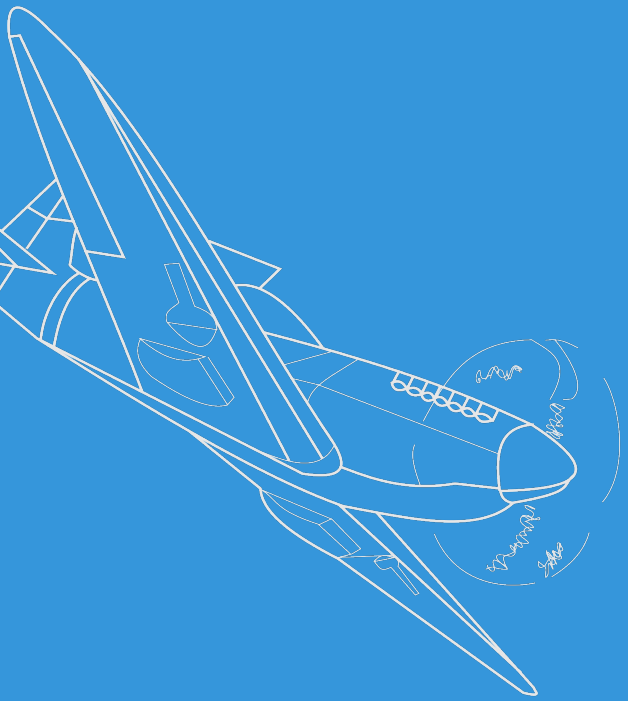
При одинаковом приоритете используется тот `location`, что находите **выше** в конфиге.

Отдача статических документов


```
location ~* ^.+\.(jpg|jpeg|gif|png)$ {
    root         /www/images;
}

location /sitemap/ {
    alias /home/www/generated/;
}
```

`/2015/10/ae2b5.png` → `/www/images/2015/10/ae2b5.png`
`/sitemap/index.xml` → `/home/www/generated/index.xml`



Сервер-приложения (application server)

Backend (application) сервер

Роль application сервера заключается в исполнении бизнес-логики приложения и генерации динамических документов.

На каждый HTTP запрос application сервер запускает некоторый обработчик в приложении. Это может быть функция, класс или программа, в зависимости от технологии.

Подробнее про различие web server и application server:
https://youtu.be/BcmUOmvl1N8

Протоколы запуска приложений

1. Servlets и др. специализированные API
2. mod_perl, mod_python, mod_php
3. CGI
4. FastCGI
5. SCGI
6. PSGI, **WSGI**, Rack

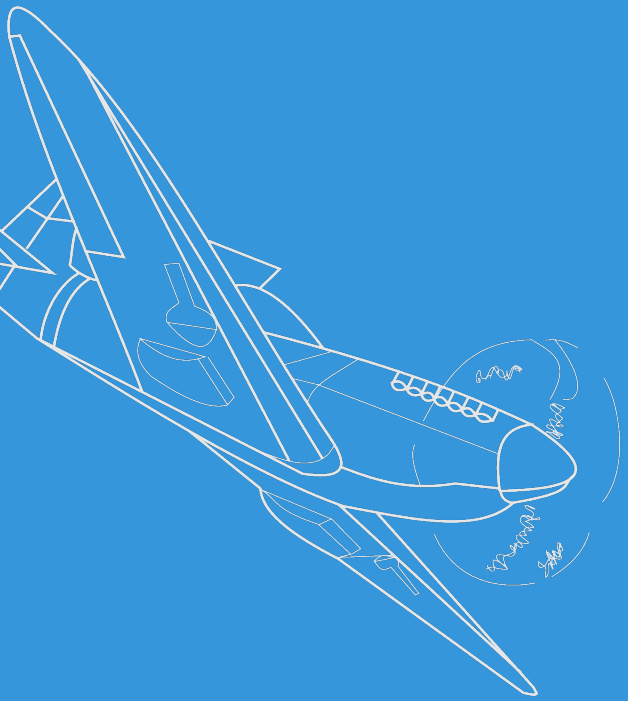
CGI — Common Gateway Interface

- Метод, QueryString, заголовки запроса — через **переменные окружения**;
- Тело запроса передаётся через **STDIN**;
- Заголовок и тело ответа возвращаются через **STDOUT**;
- HTTP-код ответа передаётся через псевдозаголовок **Status**;
- Поток ошибок **STDERR** направляется в лог ошибок сервера.

Переменные окружения CGI

- `REQUEST_METHOD` — метод запроса,
- `PATH_INFO` — путь из URL,
- `QUERY_STRING` — фрагмент URL после `?`,
- `REMOTE_ADDR` — IP-адрес пользователя,
- `CONTENT_LENGTH` — длина тела запроса,
- `HTTP_COOKIE` — Заголовок `Cookie`,
- `HTTP_ANY_HEADER_NAME` — любой другой HTTP-заголовок.

WSGI



WSGI — актуальный протокол

WSGI, PSGI, Rack — протоколы вызова функции обработчика из application сервера. Сам application server при этом может выполняться в отдельном процессе или совпадать с web сервером.

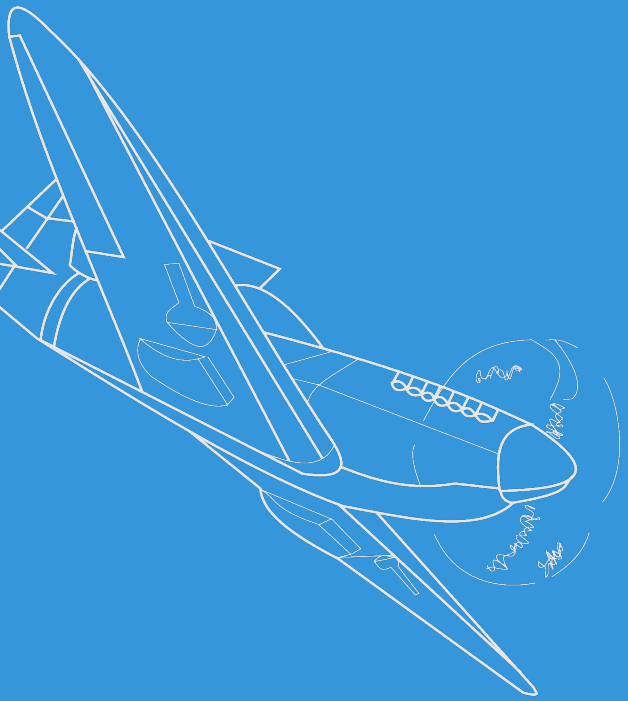
Как правило, при использовании этих протоколов в качестве application сервера выступает отдельный легковесный процесс.

Простое WSGI-приложение

1. pip install gunicorn
2. pip freeze > requirements.txt
3. cat myapp.py
4.
```
def app(environ, start_response):
    data = b"Hello, world!\n"
    start_response("200 OK", [
        ("Content-Type", "text/plain"),
        ("Content-Length", str(len(data)))
    ])
    return iter([data])
```
4. gunicorn --workers 4 myapp:app

Web Server Gateway Interface

- Обработчик — функция или класс (callable);
- Метод QueryString, заголовки запроса - через аргумент **environ**;
- Тело запроса передаётся через file-handle **wsgi.input**;
- HTTP-код ответа и заголовки ответа передаются через вызов функции **start_response**;
- Тело ответа возвращается в виде списка (*iterable*) из обработчика;
- Поток ошибок должен быть направлен в file-handle **wsgi.stderr**.



Настройка проксирования в nginx

Настройка проксирования в nginx


```
proxy_set_header Host       $host;
proxy_set_header X-Real-IP $remote_addr;
location / {
    proxy_pass http://backend;
}
location /partner/ {
    proxy_pass http://www.partner.com;
}
location ~ \.\w\w\w?\w?$ {
    root /www/static;
}
```

Настройка upstream в nginx

```
upstream backend  {
    server back1.example.com:8080 weight=1 max_fails=3;
    server back2.example.com:8080 weight=2 fail_timeout=360s;
    server unix:/tmp/backend.sock;
    server backup1.example.com:8080 backup;
    server backup2.example.com:8080 backup;
}
```

- `fail_timeout` – таймаут;
- `max_fails` – количество ошибок, после которого сервер попадёт в чёрный список.
- `weight` – вес сервера, другими словами, какую долю запросов слать на этот сервер.

Производительность


```
# wrk для Ubuntu
sudo apt-get install build-essential libssl-dev git -y
git clone https://github.com/wg/wrk.git wrk
cd wrk
make
# wrk для MacOS
brew install wrk
# Скопировать в папку, откуда окружение сможет найти бинарь
sudo cp wrk /usr/local/bin
# ab для Ubuntu
sudo apt install apache2-utils
```

Производительность (1)

Опции утилиты `wrk`:

- `-t` – количество потоков;
- `--timeout` – таймаут;
- `-d` – время выполнения;
- `-c` – количество открытых соединения на все потоки (на каждый поток – `<количество соединений>/<количество потоков>`);

Пример запуска:

```
wrk -t12 -c400 -d30s http://127.0.0.1:8080/index.html
```

Производительность (2)

Опции утилиты `ab`:

- `-n` – количество запросов;
- `-c` – количество параллельных запросов;
- `-t` – таймаут;
- `...`

Пример запуска:

```
ab -n 10 -c 2 -t 1 -v 2 http://127.0.0.1/index.html
```

Домашнее задание № 2

1. Установить `nginx` и `gunicorn` — 2 балла;
2. Настроить `nginx` для отдачи статический файлов из `public/` — 2 балла;
3. Создать простейшее `WSGI`-приложение и запустить его с помощью `gunicorn` — 2 балла;
4. Настроить проксирование запросов на `nginx` — 2 балла;
5. Измерить производительность `nginx` и `gunicorn` с помощью `ab` или `wrk` — 2 балла.

Рекомендуемая литература

[Документация nginx]
[RFC URI]

Для саморазвития (опционально)
[Чтобы не набирать двумя пальчиками]

Спасибо за внимание!

Антон Кухтичев

a.kukhtichev@mail.ru

@toshunster