OTUS

Administrator Linux. Professional

Динамический веб

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Преподаватель

Лавлинский Николай

Технический директор «Метод Лаб»

Более 15 лет в веб-разработке

Преподавал в ВУЗе более 10 лет
Более 5 лет в онлайн-образовании

Специализация: оптимизация производительности, ускорение сайтов и веб-приложений
https://t.me/methodlab_tg
https://www.methodlab.ru/
https://rutube.ru/channel/24617406/
https://www.youtube.com/c/NickLavlinsky
https://www.youtube.com/@site_support
https://vk.com/nick.lavlinsky

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в Телеграм-чате

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

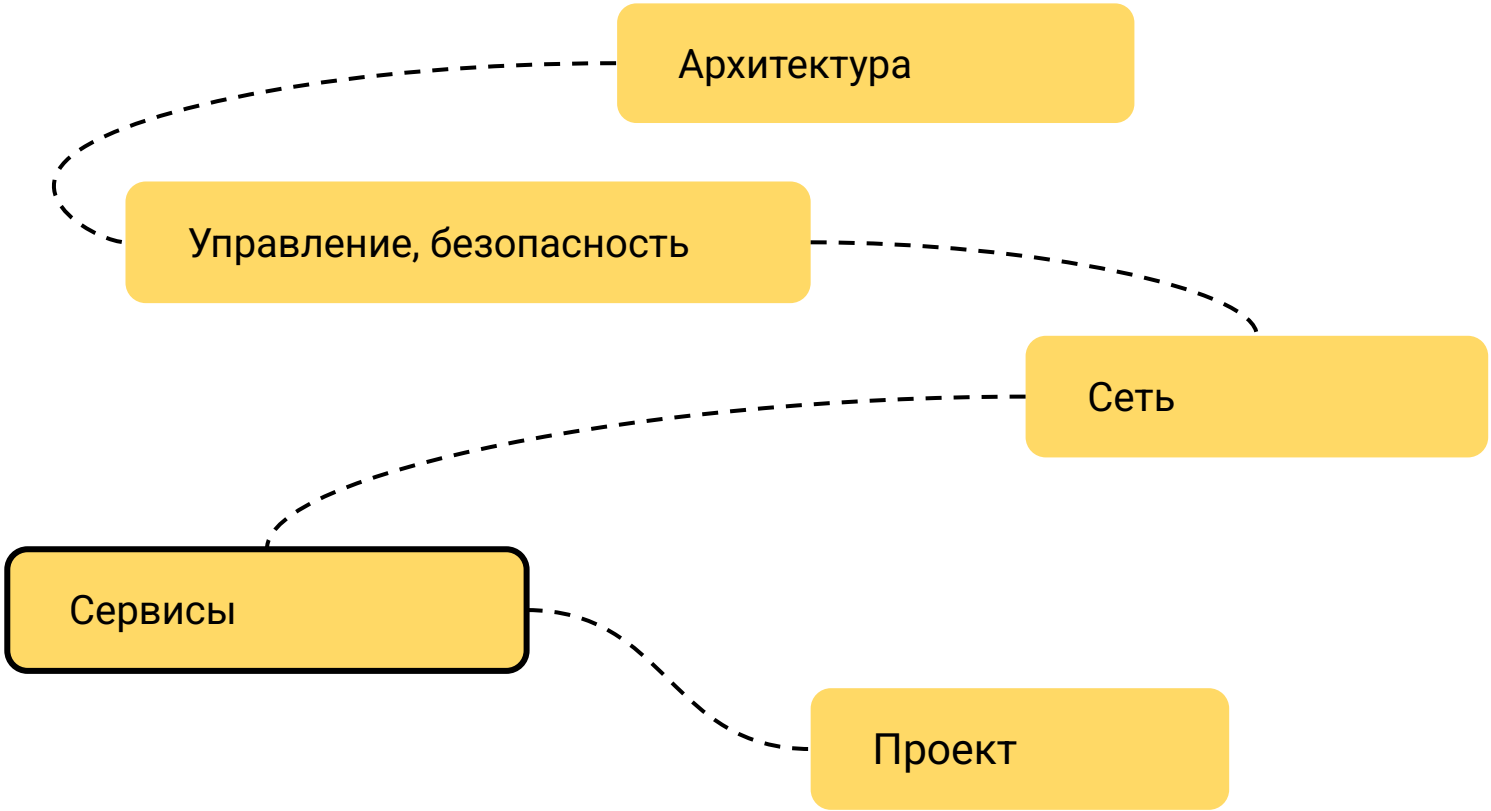
Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Карта курса


Архитектура
Управление, безопасность
Сеть
Сервисы
Проект

Маршрут вебинара

Цели вебинара

После занятия вы сможете

1. Понимать варианты взаимодействия компонентов веб-приложения
2. Выбирать эффективную архитектуру
3. Различать методы обработки соединений

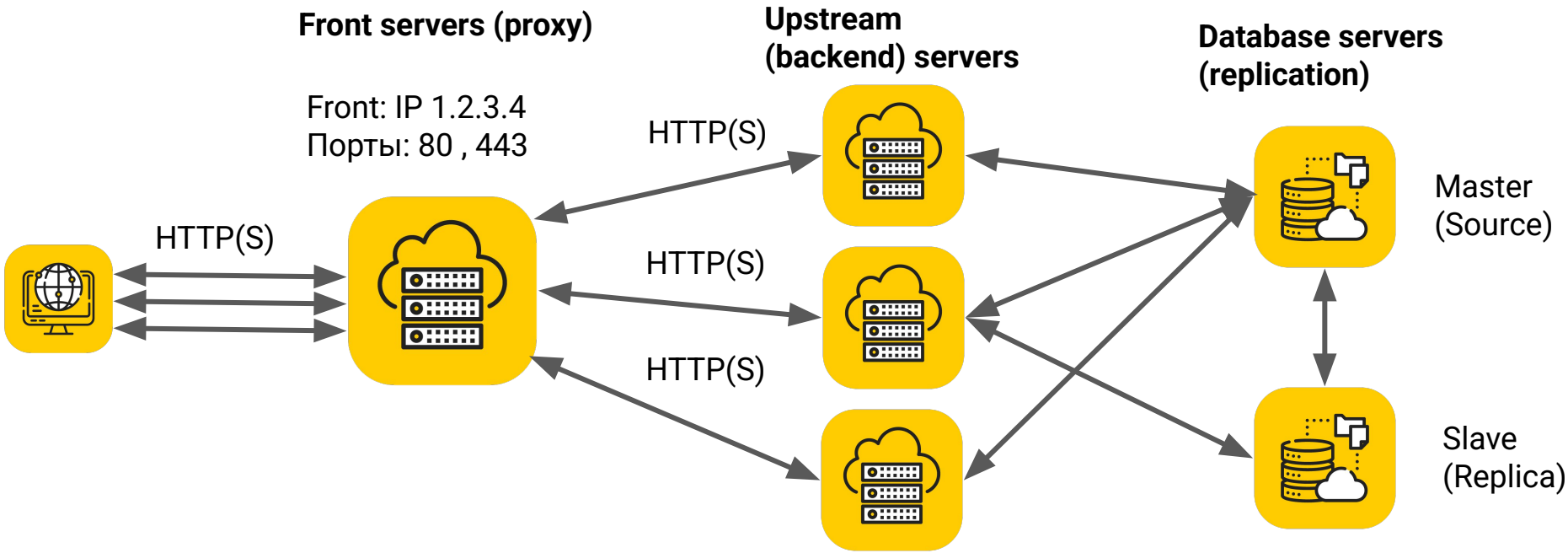
Смысл

Зачем вам это уметь

1. Запускать веб-приложения
2. Решать проблемы и отлаживать системы
3. Оптимизировать производительность

Архитектура веб-приложений

Стандартная архитектура приложения



Протоколы взаимодействия

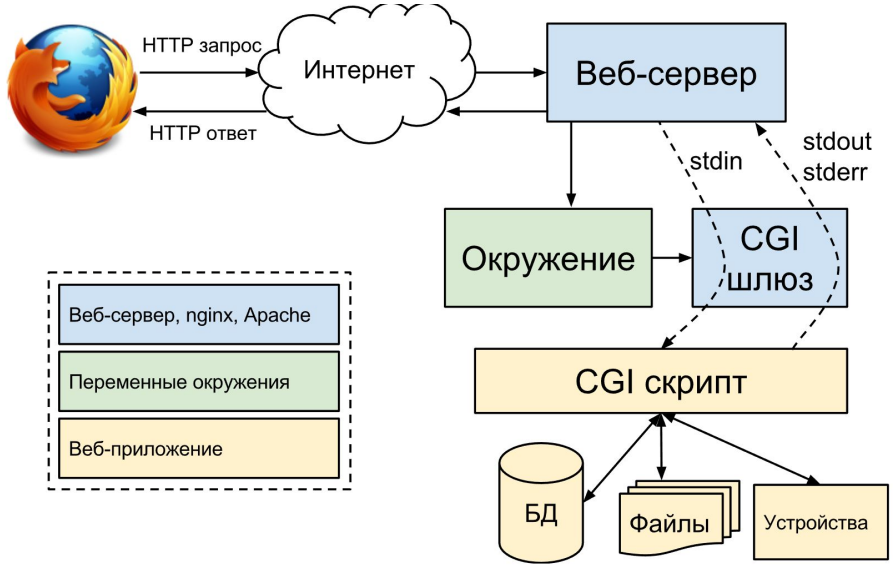
- Протоколы взаимодействия с сервером приложений
    - CGI
    - FastCGI
    - SCGI
    - WSGI
- Клиентские протоколы (браузер-сервер)
    - AJAX (XHR)
    - WebSocket

https://unit.nginx.org/

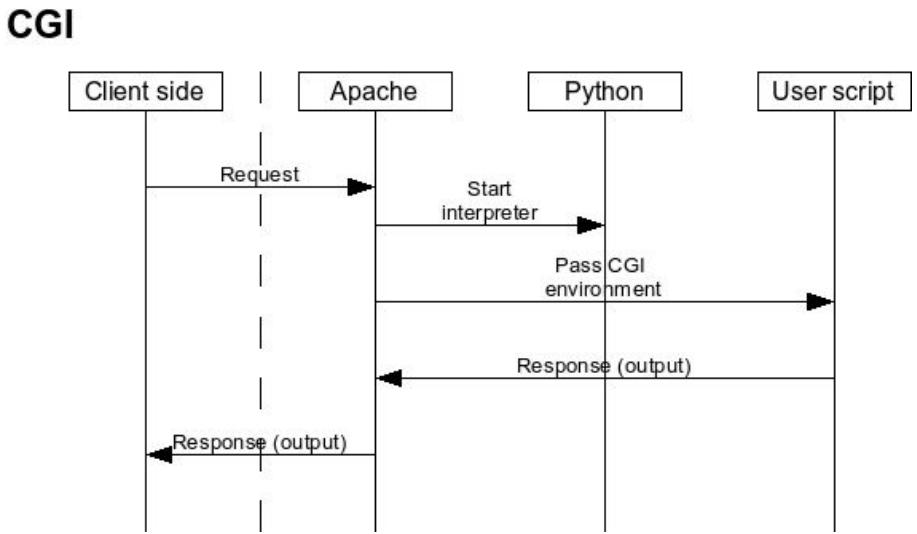
Протоколы взаимодействия с сервером приложений

CGI

- CGI (Common Gateway Interface) - стандарт интерфейса, используемого для связи внешней программы с веб-сервером
- Скрипт запускается при каждом запросе
- CGI-параметры передаются в виде переменных окружения
- Скрипт возвращает результат в STDOUT, который транслируется сервером клиенту
- Высокие накладные расходы на интерпретацию кода (каждый запрос)



CGI — процесс обработки запроса



CGI — конфигурация (Apache)

```
LoadModule cgi_module /usr/lib/apache2/modules/mod_cgi.so
AddHandler cgi-script .cgi .pl

ScriptAlias /cgi-bin/ /home/www/cgi-bin/
```

https://httpd.apache.org/docs/2.4/howto/cgi.html

FastCGI

- Протокол взаимодействия между веб-сервером и сервером приложений
- Блокирующий доступ
- Используется TCP или UNIX socket
- Пример конфигурации в Nginx:

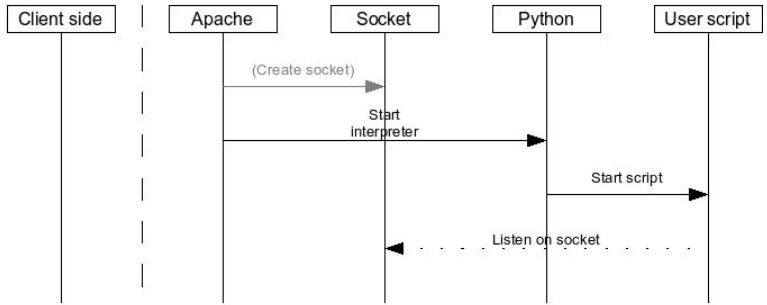
```
location ~ ^/(status|ping)$ {
    allow 127.0.0.1;
    deny all;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_index index.php;
    include fastcgi_params;
    fastcgi_pass unix:/var/run/phpfpm-api.sock;
}
```

FastCGI — принципы работы

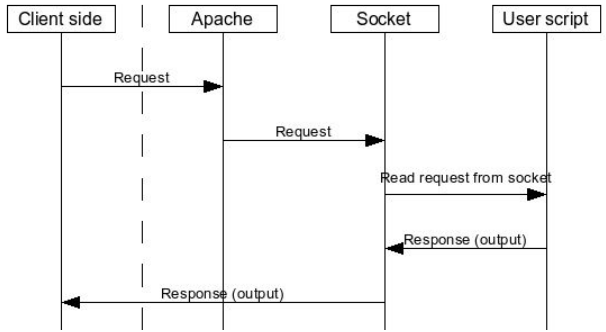
- Программа загружается в память в качестве при первом обращении к серверу
- Не требуется процесс интерпретации и подготовки кода к исполнению
- Один и тот же процесс обрабатывает множество запросов
- Высокая производительность
- Большее потребление оперативной памяти

FastCGI — процесс обработки запроса

FastCGI (Startup)

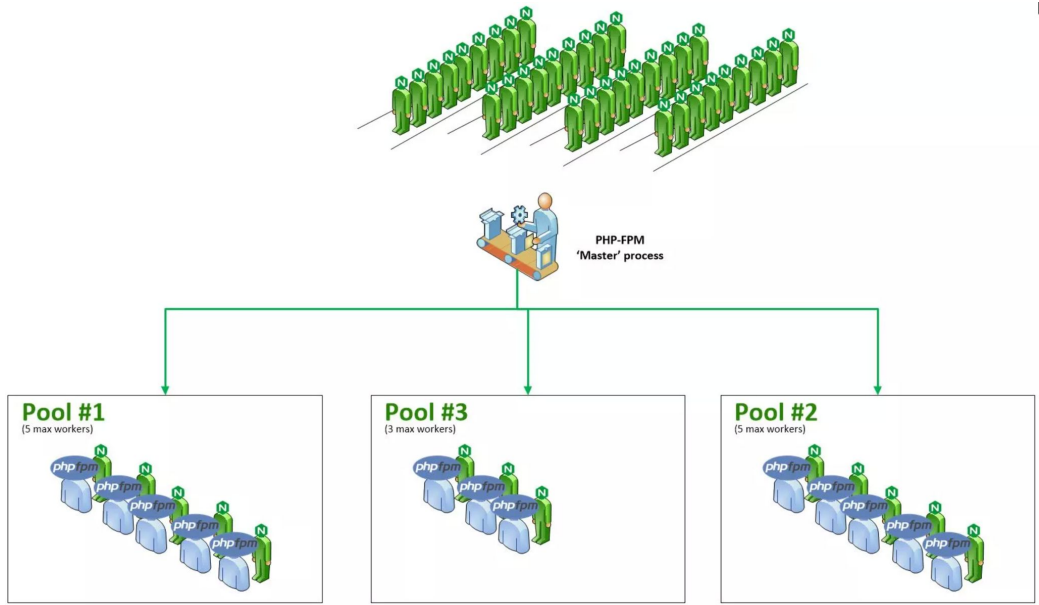


FastCGI (Request handling)



PHP-FPM

- Популярная реализация FastCGI-сервера для PHP
- Работает с большинством веб-приложений
- Позволяет настраивать пулы для приложений
- Статическое и динамическое количество воркеров

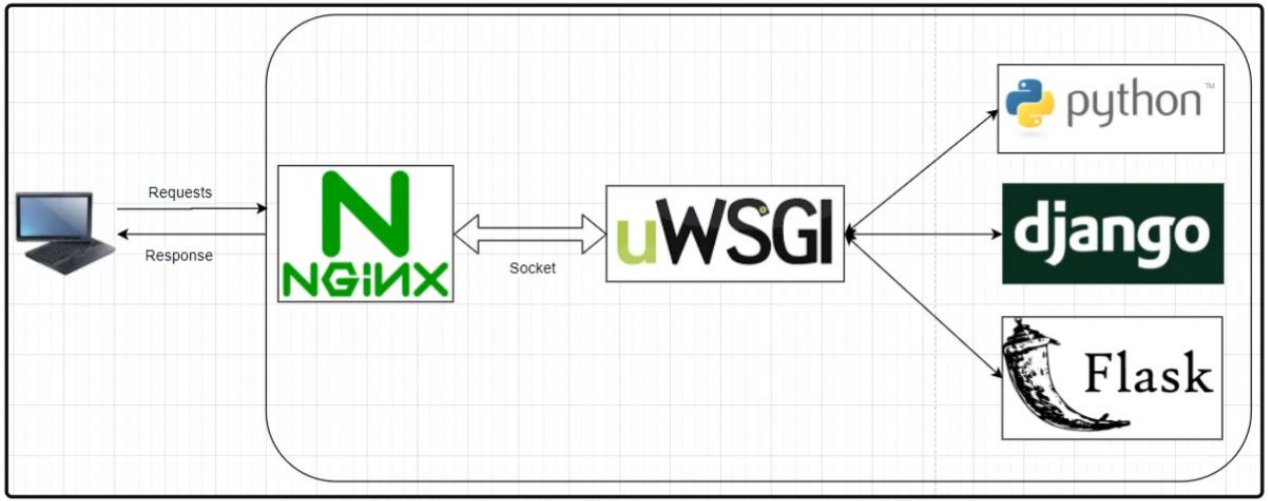


WSGI framework

- Стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером
- Должно быть вызываемым (callable) объектом (обычно это функция или метод)
- Параметры на входе
    - Словарь переменных окружения (environ)
    - Обработчик запроса (start_response)
- Вызывает обработчик запроса с кодом HTTP-ответа и HTTP-заголовками
- Возвращает итерируемый объект с телом ответа

uWSGI

- uWSGI — веб-сервер и сервер веб-приложений, первоначально реализованный для запуска приложений Python через протокол WSGI (и его бинарный вариант uwsgi)
- Версия 2.0 поддерживает также запуск веб-приложений Lua, Perl, Ruby и других



uWSGI

```
location / {
        include         uwsgi_params;
        uwsgi_pass      localhost:81;
    }
```

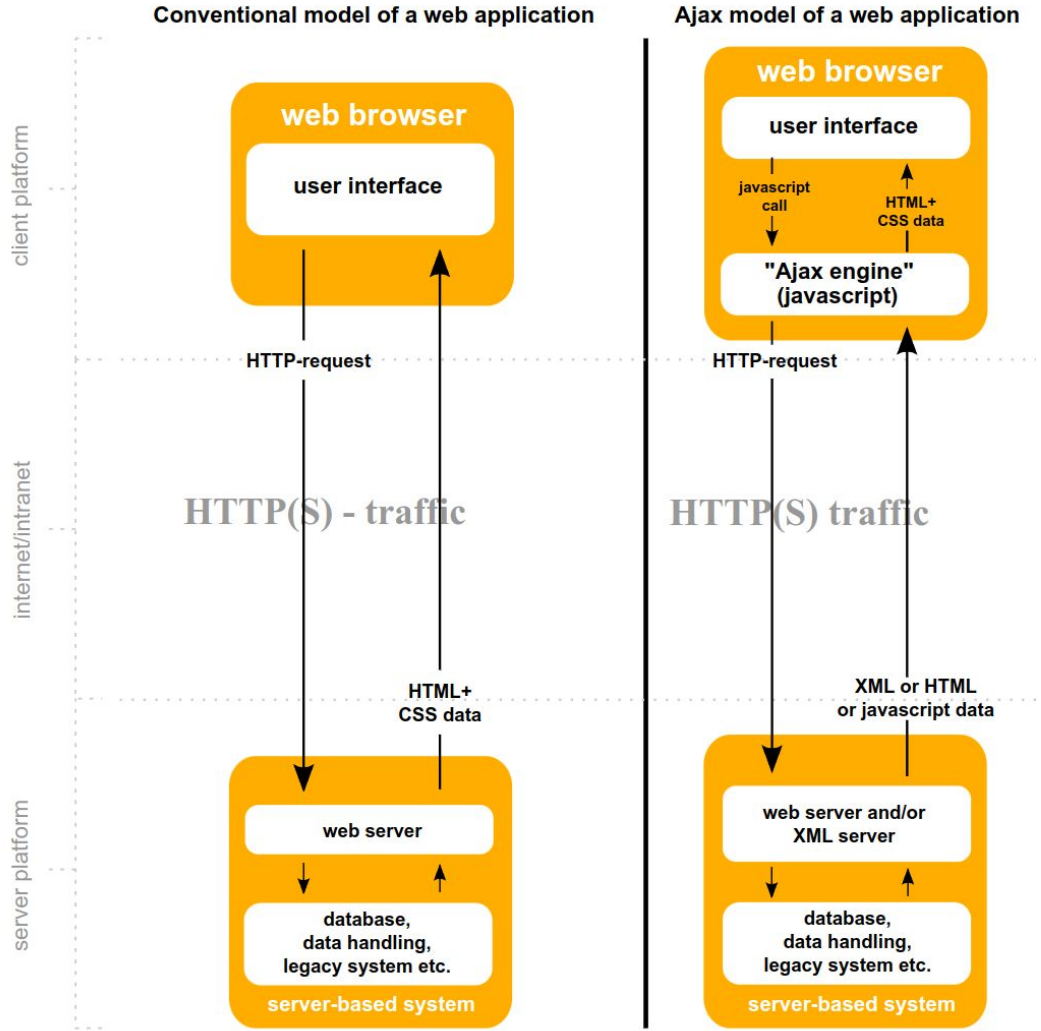
http://nginx.org/en/docs/http/ngx_http_uwsgi_module.html

Клиентские протоколы (браузер-сервер)

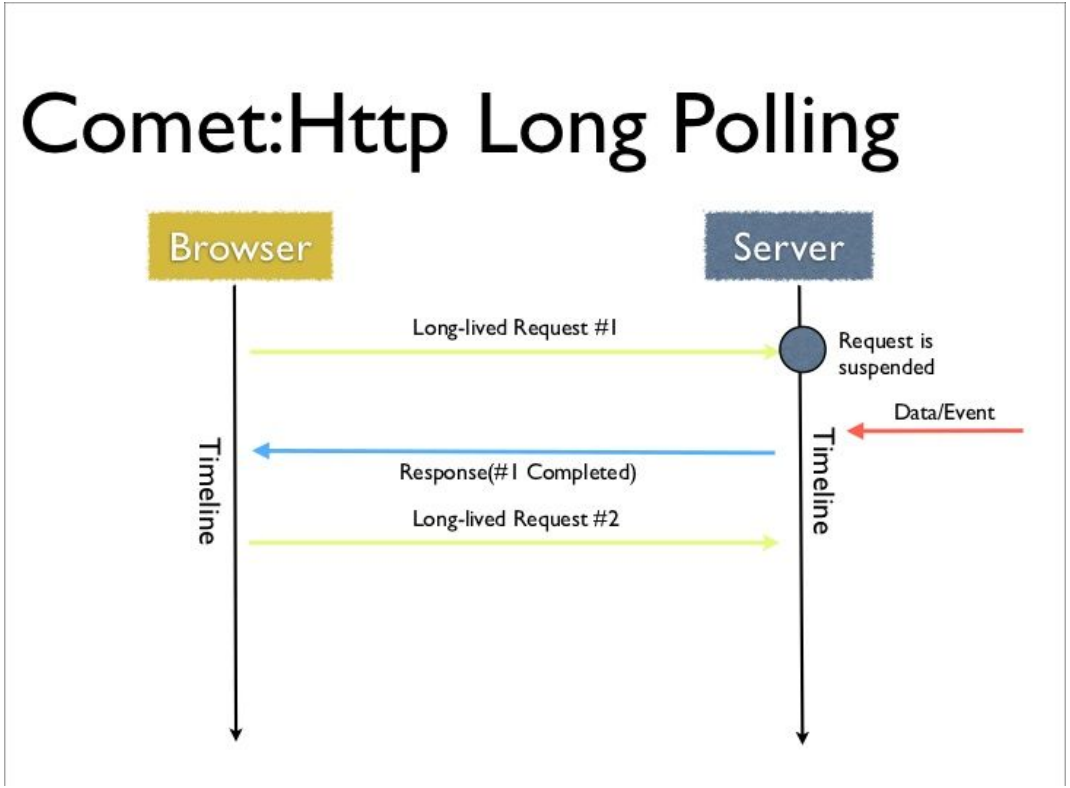
AJAX (XHR)

- Asynchronous Javascript and XML — «асинхронный JavaScript и XML»
- XMLHttpRequest (XMLHTTP, XHR) — API в JS для отправки HTTP-запросов
- fetch — современная версия API в JS
- Отправка запросов без перезагрузки страницы
- Обновление частей страницы
- Динамическая загрузка (бесконечная прокрутка)
- Данные в формате JSON или HTML

AJAX


Conventional model of a web application
Ajax model of a web application
client platform
internet/intranet
server platform
web browser
user interface
HTTP-request
HTTP(S) - traffic
HTML+
CSS data
web server
database,
data handling,
legacy system etc.
server-based system
web browser
user interface
javascript
call
HTML+
CSS data
"Ajax engine"
(javascript)
HTTP-request
HTTP(S) traffic
XML or HTML
or javascript data
web server and/or
XML server
database,
data handling,
legacy system etc.
server-based system

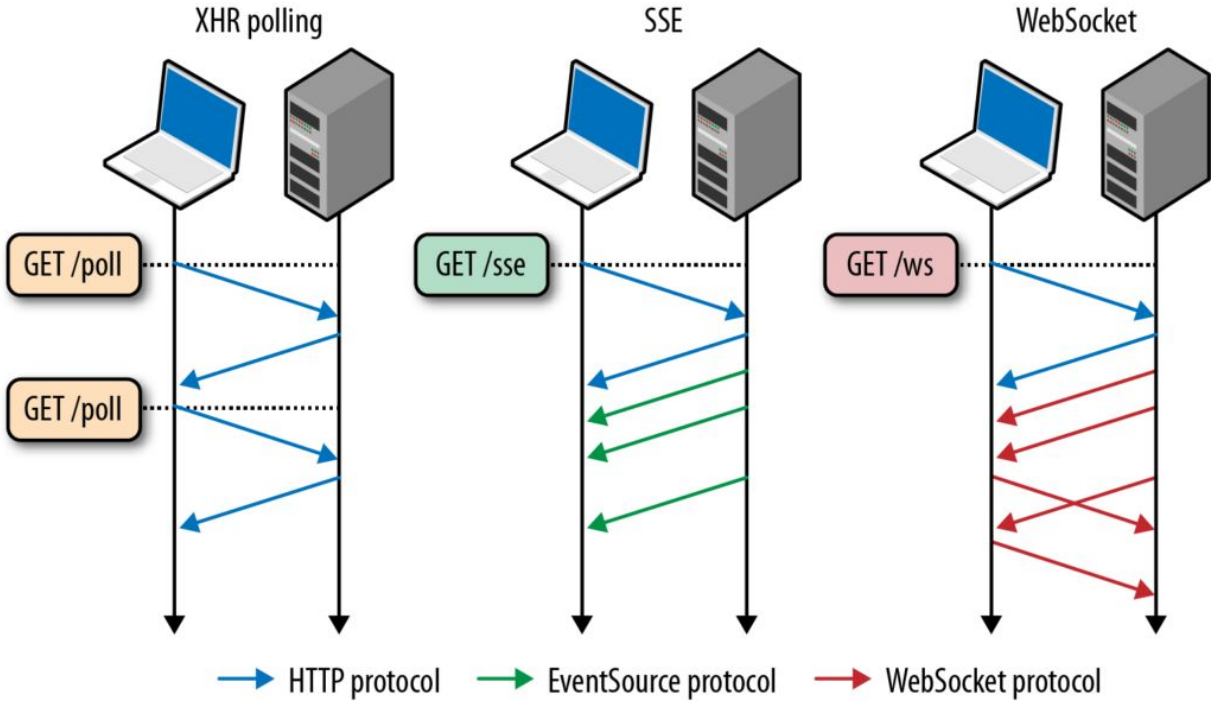
AJAX Long Polling



WebSocket

- Протокол для долгоживущих соединений поверх TCP
- Соединение устанавливается с помощью HTTP
- Сообщения могут идти в обе стороны
- Схема URL:
    - HTTP: ws://ws.my.site
    - HTTPS: wss://wss.my.site

WebSocket vs XHR vs SSE (Server-sent events)



WebSocket в Nginx

```
http {
    map $http_upgrade $connection_upgrade {
        default upgrade;
        '' close;
    }
…
location /wsapp/ {
      proxy_pass http://wsbackend;
      proxy_http_version 1.1;
      proxy_set_header Upgrade $http_upgrade;
      proxy_set_header Connection $connection_upgrade;
      proxy_set_header Host $host;
}
```

https://www.nginx.com/blog/websocket-nginx/

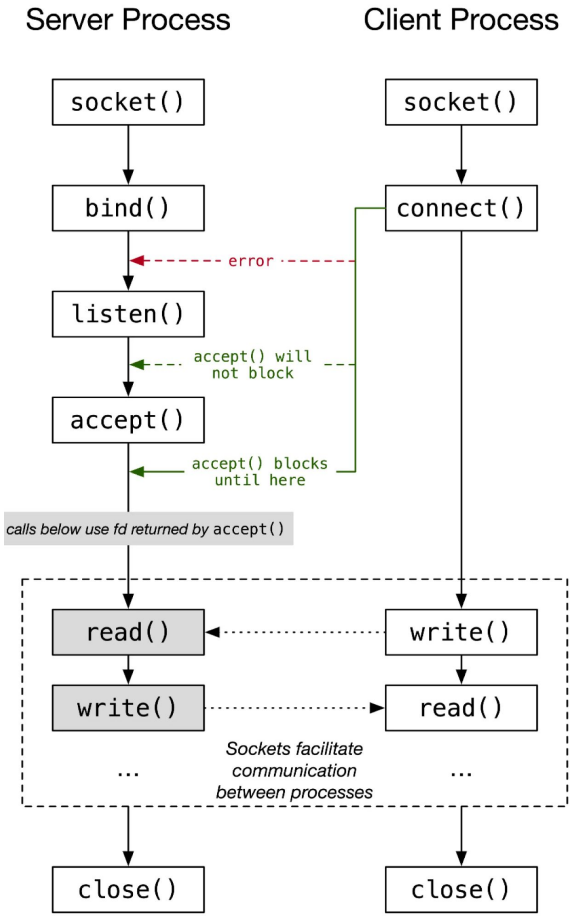
Методы обработки соединений

Обработка соединений

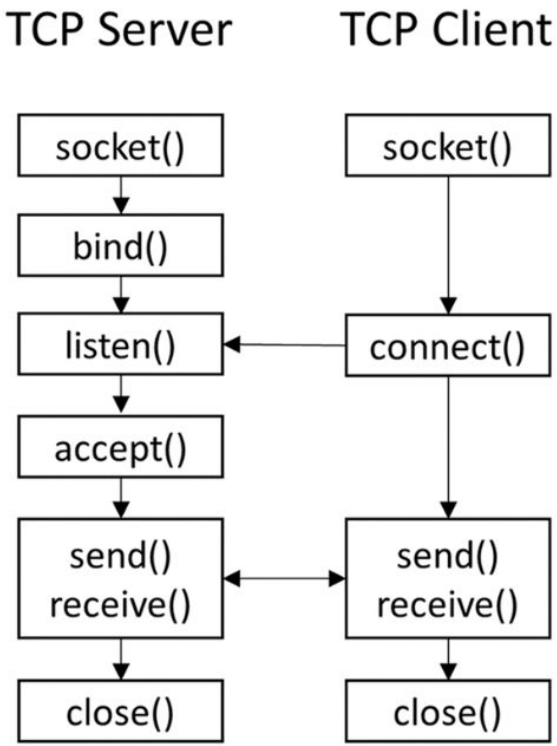
- Один клиент
    - UNIX-сокет
    - TCP-сокет
- Мультиплексирование
    - `select()`
    - `epoll()`

http://nginx.org/ru/docs/events.html

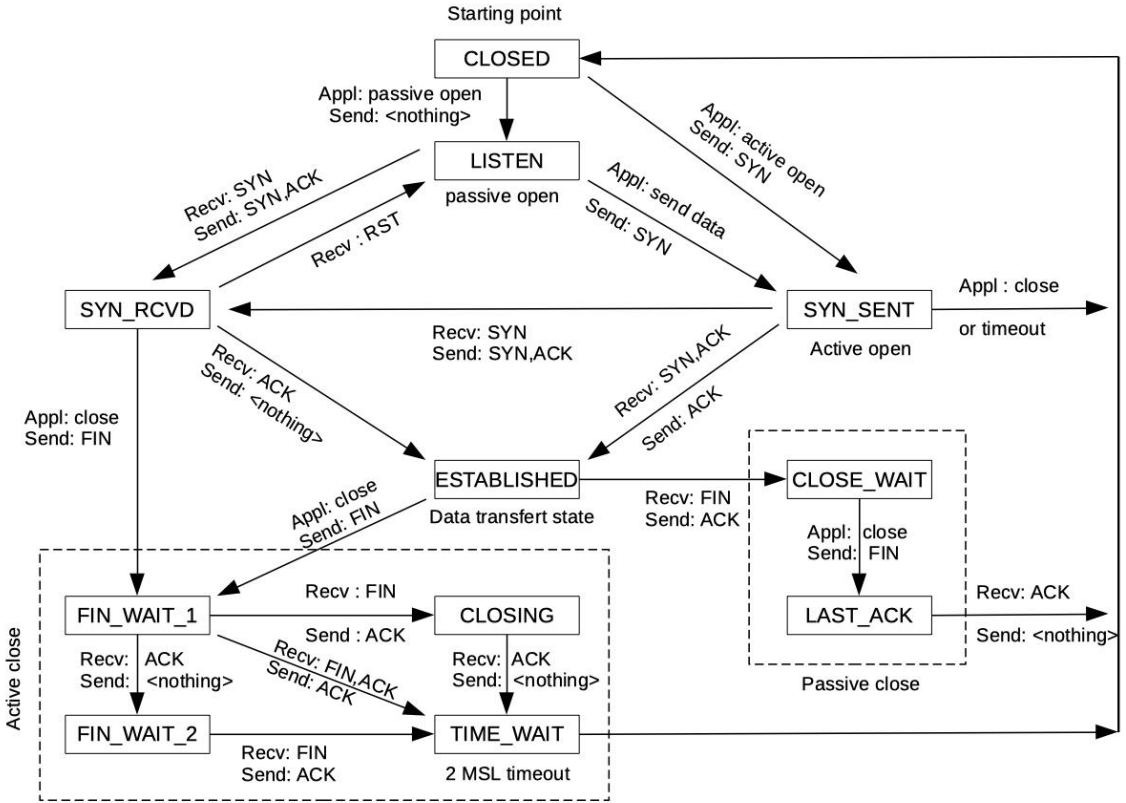
Unix-сокет


Server Process
Client Process
socket()
socket()
bind()
connect()
error
listen()
accept() will not block
accept()
accept() blocks until here
calls below use fd returned by accept()
read()
write()
write()
read()
...
Sockets facilitate communication between processes
...
close()
close()

TCP-сокет


TCP Server
TCP Client
socket()
socket()
bind()
listen()
connect()
accept()
send()
receive()
send()
receive()
close()
close()

Состояния TCP-сокета



Обработка соединений select()

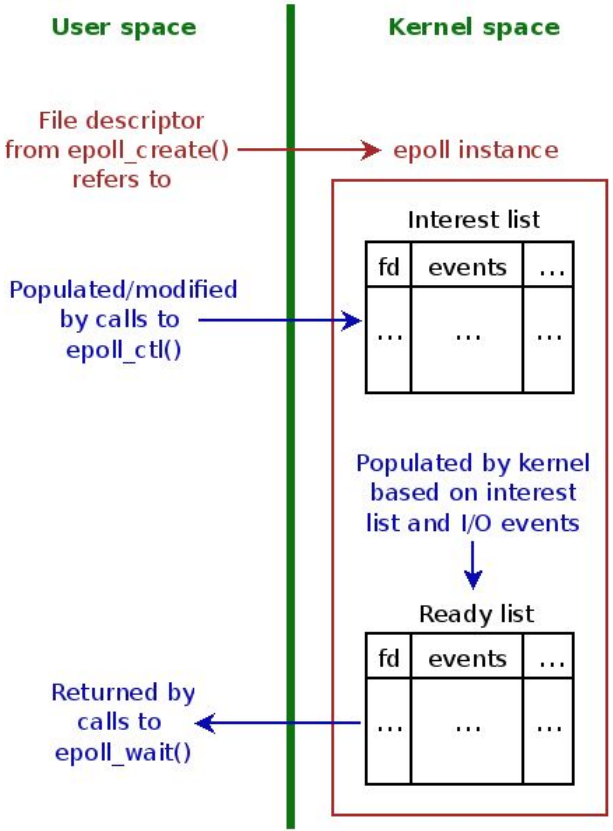
- На каждом вызове опрашиваются все файловые дескрипторы на предмет изменения статуса
- Ограниченный набор дескрипторов (select())
- Псевдокод обработки с select()

```
select(socket);
while (1) {
     sockets = select();
     for (socket in sockets) {
          if (can_read(socket)) {
               read(socket, buffer);
               process(buffer);
          }
     }
}
```

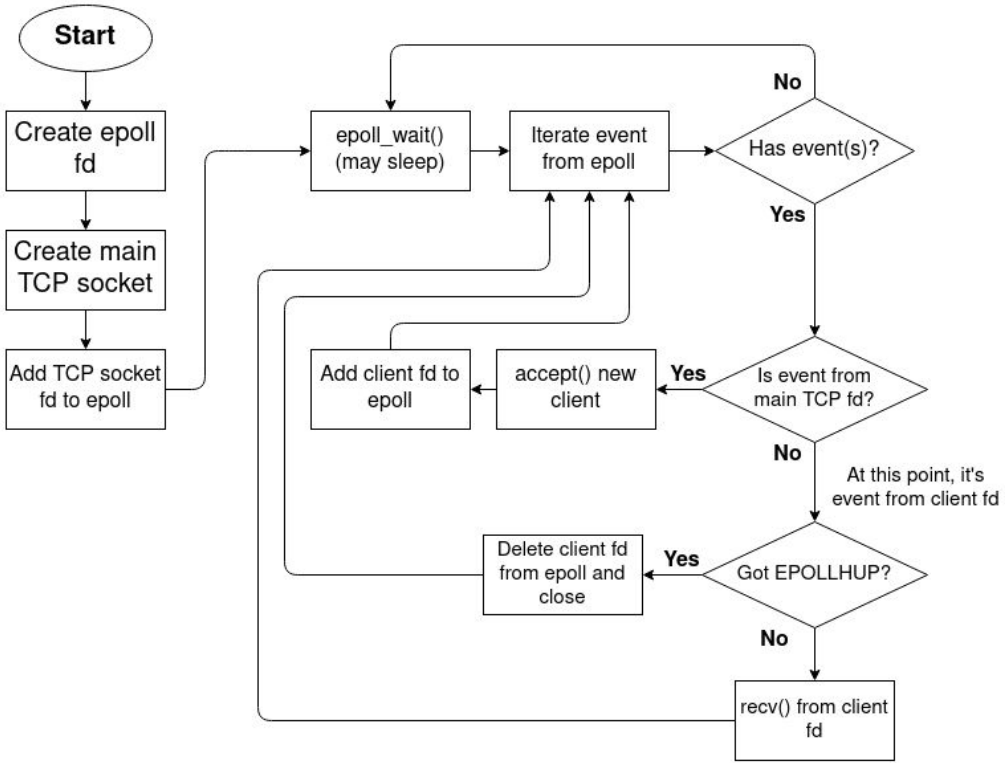
Обработка соединений epoll()

- API мультиплексированного ввода-вывода
- Позволяет приложениям осуществлять мониторинг нескольких открытых файловых дескрипторов
- Более эффективная замена вызовам select() и poll(), определёнными в POSIX
- Может предоставить более эффективный механизм для приложений, обрабатывающих большое количество одновременно открытых соединений — со сложностью O(1)
- /proc/sys/fs/epoll/max_user_watches - максимальное количество наблюдаемых fd для каждого RealUID

epoll()


User space
Kernel space
File descriptor
from epoll_create()
refers to
epoll instance
Interest list
fd
events
...
Populated/modified
by calls to
epoll_ctl()
Populated by kernel
based on interest
list and I/O events
Ready list
Returned by
calls to
epoll_wait()

epoll()


Start
Create epoll fd
Create main TCP socket
Add TCP socket fd to epoll
epoll_wait() (may sleep)
Iterate event from epoll
Has event(s)?
No
Yes
Is event from main TCP fd?
Yes
accept() new client
Add client fd to epoll
No
At this point, it's event from client fd
Got EPOLLHUP?
Yes
Delete client fd from epoll and close
No
recv() from client fd

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Практика

Домашнее задание

1. Создать стенд с рабочими веб-приложениями на 3 разных стеках, например:
   - Nginx + PHP-FPM (wordpress/laravel) + Python (flask/django) + JS (node.js)
   - Nginx + Java (tomcat/jetty/netty) + Go + Ruby (Ruby on rails)
2. Реализации на выбор:
   - на хостовой системе через конфиги в /etc;
   - деплой через docker-compose.
3. Описать процесс запуска и тестирования в README репозитория.

В чат ДЗ отправьте ссылку на ваш git-репозиторий.

Сроки выполнения: указаны в личном кабинете

Что мы изучили?

Подведем итоги

1. Архитектуру веб-приложения
2. Варианты взаимодействия между серверами
3. Клиентские протоколы в вебе
4. Методы обработки соединений

Список материалов для изучения

1. https://habr.com/ru/articles/259403/
2. http://nginx.org/ru/docs/events.html
3. https://habr.com/ru/articles/416669/
4. https://www.nginx.com/blog/websocket-nginx/
5. https://www.ibm.com/support/pages/flowchart-tcp-connections-and-their-definition
6. https://www.digitalocean.com/community/tutorials/how-to-install-wordpress-with-docker-compose-ru
7. https://unit.nginx.org/

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

Цели вебинара

Проверка достижения целей

1. Понимать варианты взаимодействия компонентов веб-приложения
2. Выбирать эффективную архитектуру
3. Различать методы обработки соединений

Рефлексия

Что было самым полезным на занятии?

Как будете применять на практике то, что узнали на вебинаре?

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Лавлинский Николай

Технический директор “Метод Лаб”

https://t.me/methodlab_tg
https://www.methodlab.ru/
https://rutube.ru/channel/24617406/
https://www.youtube.com/c/NickLavlinsky
https://www.youtube.com/@site_support
https://vk.com/nick.lavlinsky