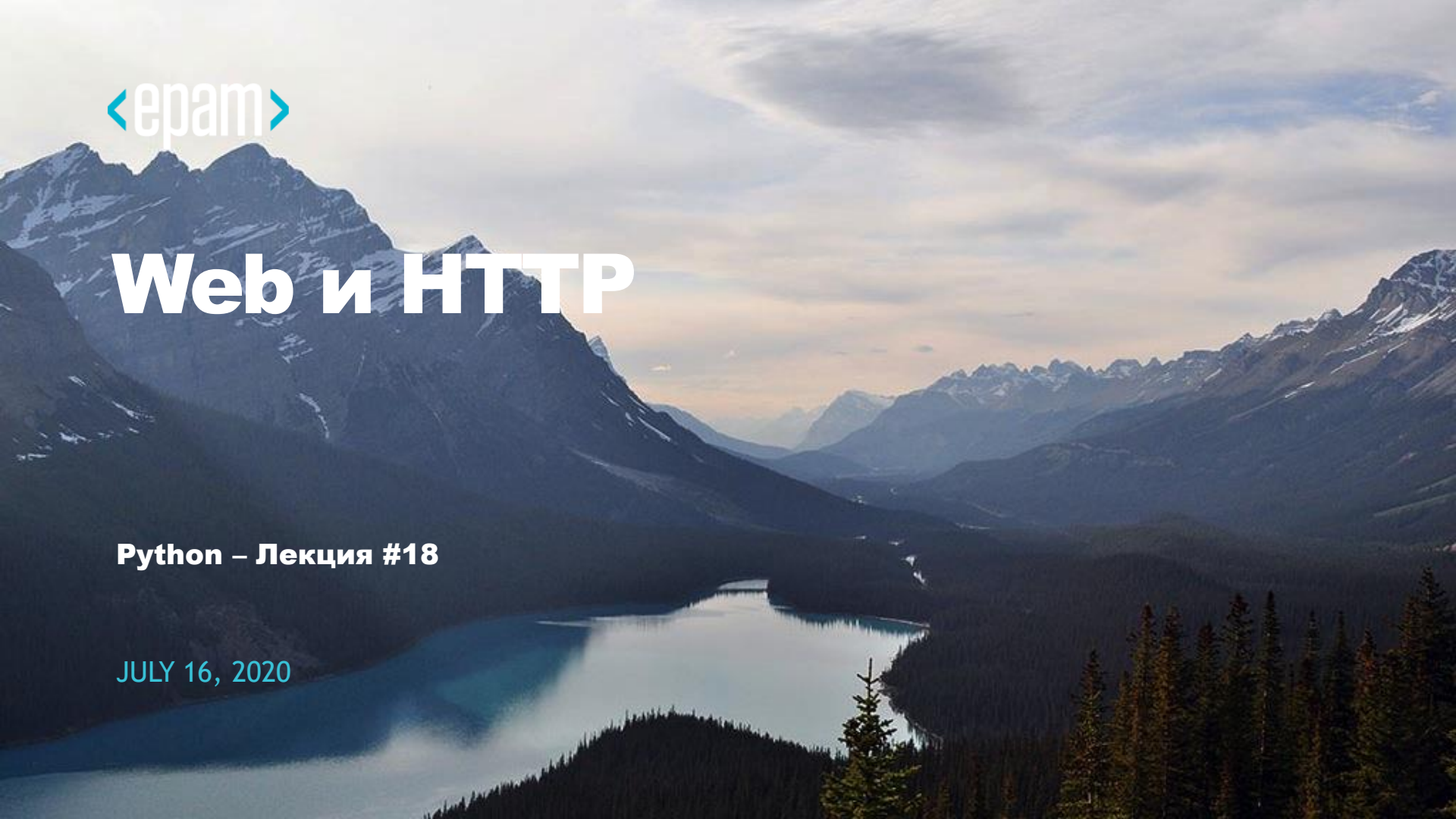



# Web и HTTP

**Python – Лекция #18**

JULY 16, 2020

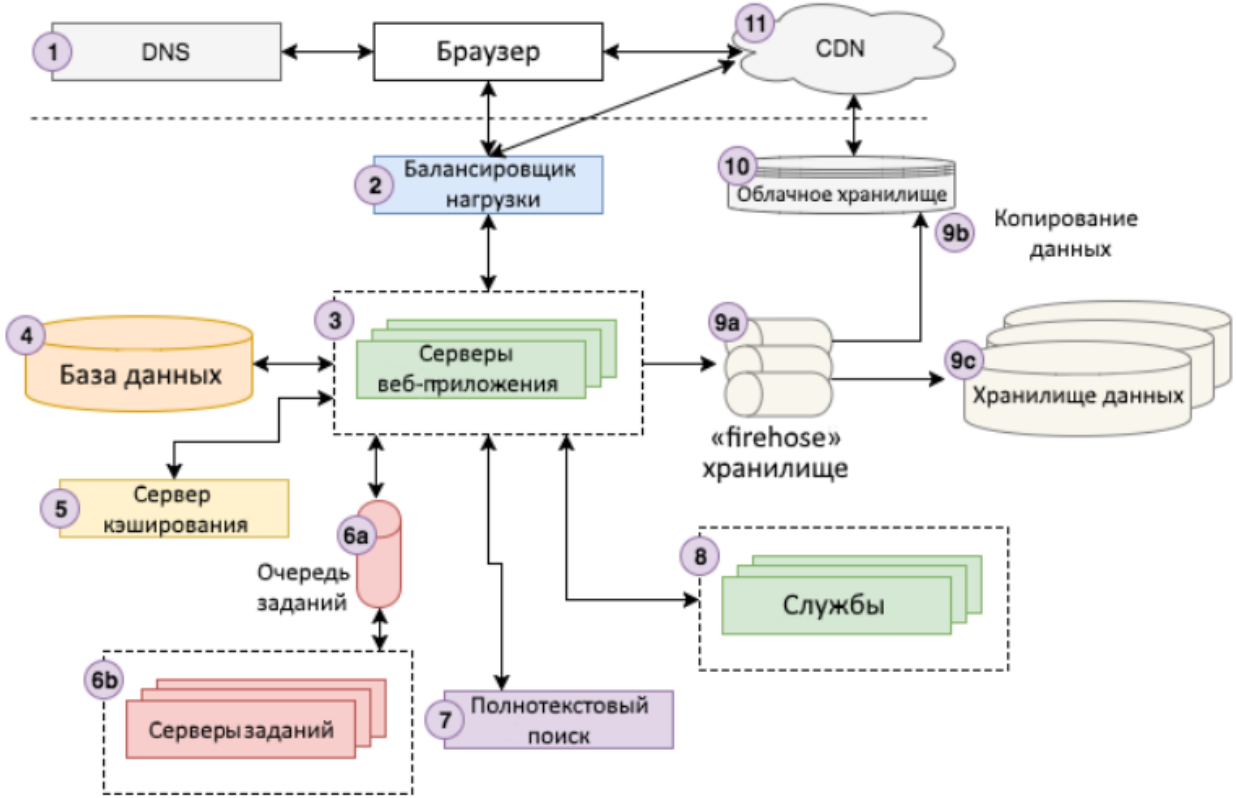
# ТЕМЫ

1. Архитектура современного WEB-а
2. Протокол HTTP, requests
3. Первое Web-приложение, Flask

Часть 1

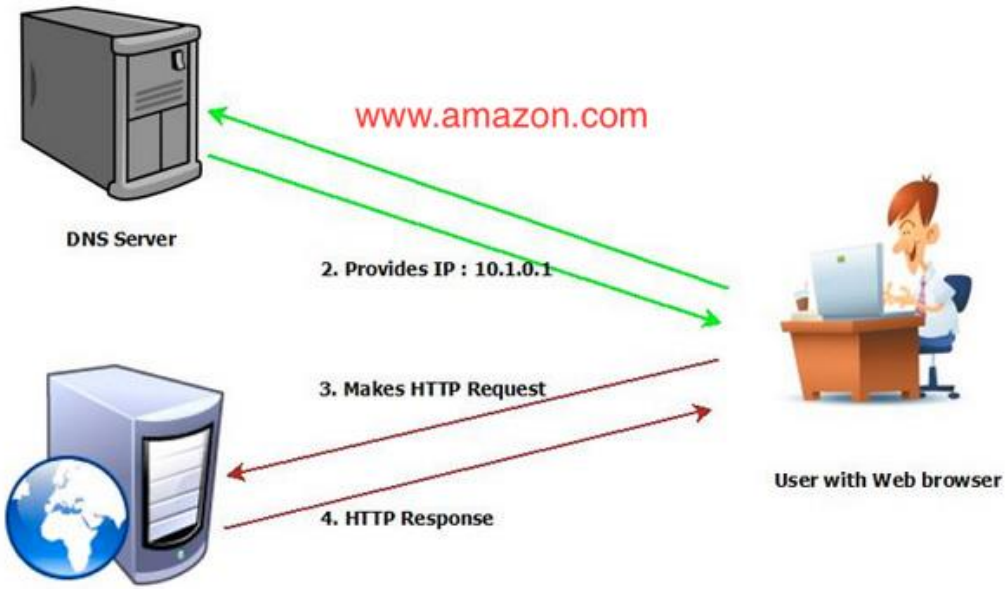
# АРХИТЕКТУРА WEB

# Архитектура современного Web приложения



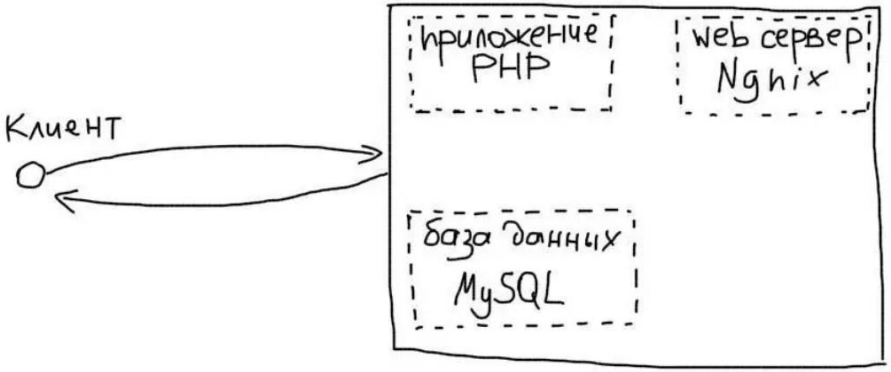
# 1. DNS (Domain Name System)

- Domain Name System (система доменных имён)
- Базовая технология, которая делает возможной работу интернета (“телефонная книга интернета”)
- DNS обеспечивает поиск пары из доменного имени и IP-адреса, что позволяет компьютеру отправить запрос на соответствующий сервер



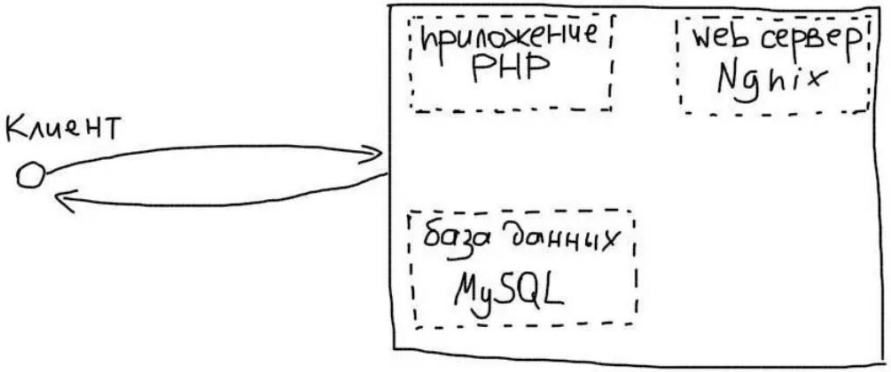
# Масштабирование Web-приложения

- Масштабирование любого Web приложения — это постепенный процесс, который включает:
  - Анализ нагрузки
  - Определение наиболее подверженных нагрузке участков
  - Вынесение таких участков на отдельные узлы и их оптимизация
  - Повтор пункта 1



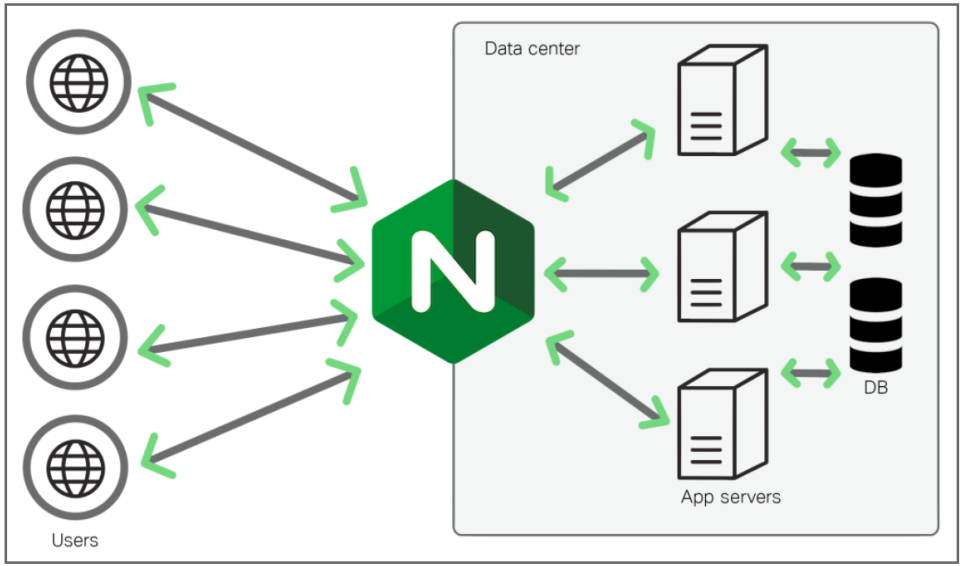
# Масштабирование Web-приложения

- Масштабирование любого Web приложения — это постепенный процесс, который включает:
  - Анализ нагрузки
  - Определение наиболее подверженных нагрузке участков
  - Вынесение таких участков на отдельные узлы и их оптимизация
  - Повтор пункта 1
- Масштабирование:
  - Вертикальное (+ CPU, HDD)
  - Горизонтальное (+ новая машина)



# 2. Балансировщик нагрузки

- Балансировщик занимается распределением запросов
- Благодаря ему возможно горизонтальное масштабирование

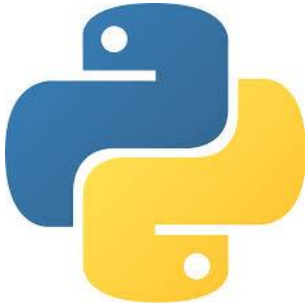


# WEB-СЕРВЕР

Веб-сервер — сервер, принимающий HTTP-запросы от клиентов и выдающий им HTTP-ответы. В зависимости от запроса сервер может генерировать ответ самостоятельно (например для статического контента), передавать запрос на обработку *серверу приложений* (англ. *Application server*) или другим образом обрабатывать их (в соответствии с настройкой).


# 3. Серверы веб-приложений (Application server)

- Принимают запросы
- Выполняют бизнес логику
- Отправляют ответы
- В нашем случае, написанное на Python
- Существуют Frameworks

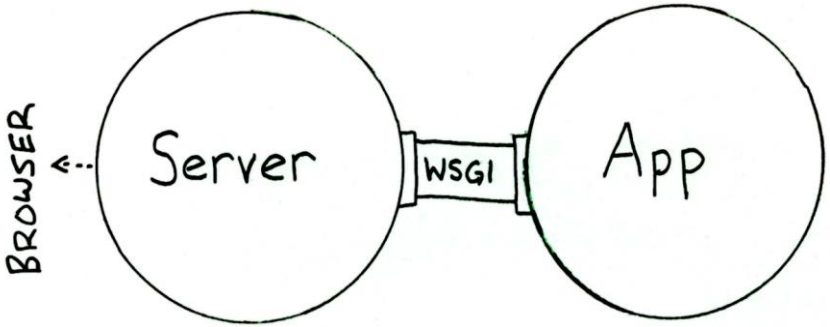




# WSGI (Web-Server Gateway Interface)


В начале 2000-х годов

WSGI-серверы появились потому, что веб-серверы в то время не умели взаимодействовать с приложениями, написанными на языке Python



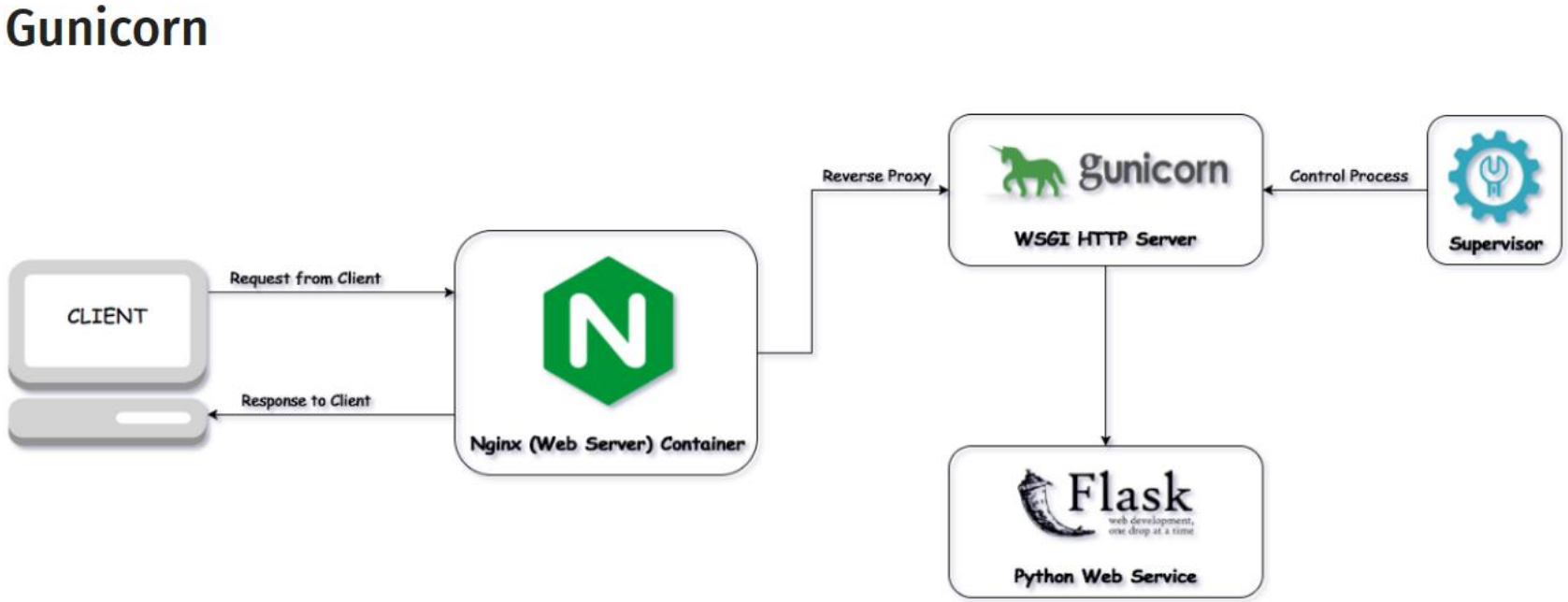
# Хорошо, но почему именно WSGI?

- **WSGI**-сервера были разработаны чтобы обрабатывать множество запросов одновременно
- **WSGI** ускоряет разработку веб-приложений написанных на языке Python
- **WSGI** дает Вам гибкость в изменении компонентов веб-стека без изменения приложения, которое работает с **WSGI**

uWSGI


ACGI

# Получаем такую протейшую схему



# 4. Сервер баз данных SQL и NoSQL





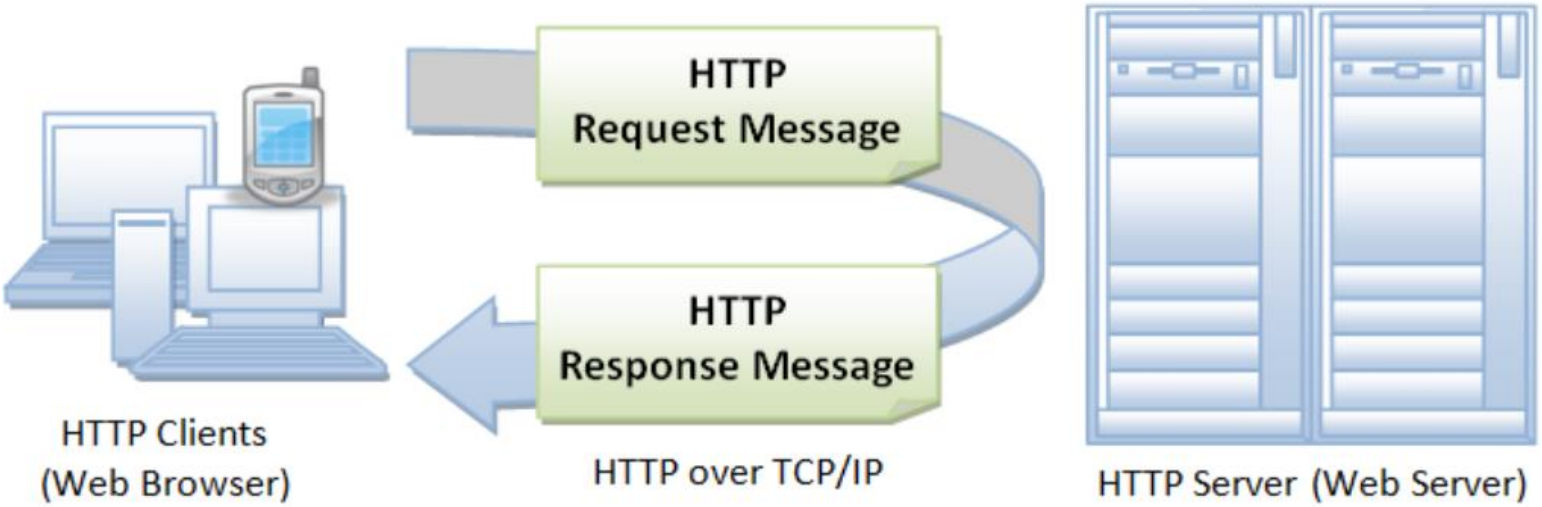

# ВОПРОСЫ?

Часть 1

# ПРОТОКОЛ HTTP

# ПРОТОКОЛ HTTP

**HTTP** (англ. *HyperText Transfer Protocol*) — протокол прикладного уровня передачи данных (изначально — гипертекстовых документов в формате «HTML», в настоящее время — произвольных данных).



# Протокол HTTP и Python

1. Библиотеки для работы с HTTP:
   - Httplib
   - Httplib2
   - Urllib
   - Treq
   - Requests (HTTP For Humans) – наиболее используемая людьми

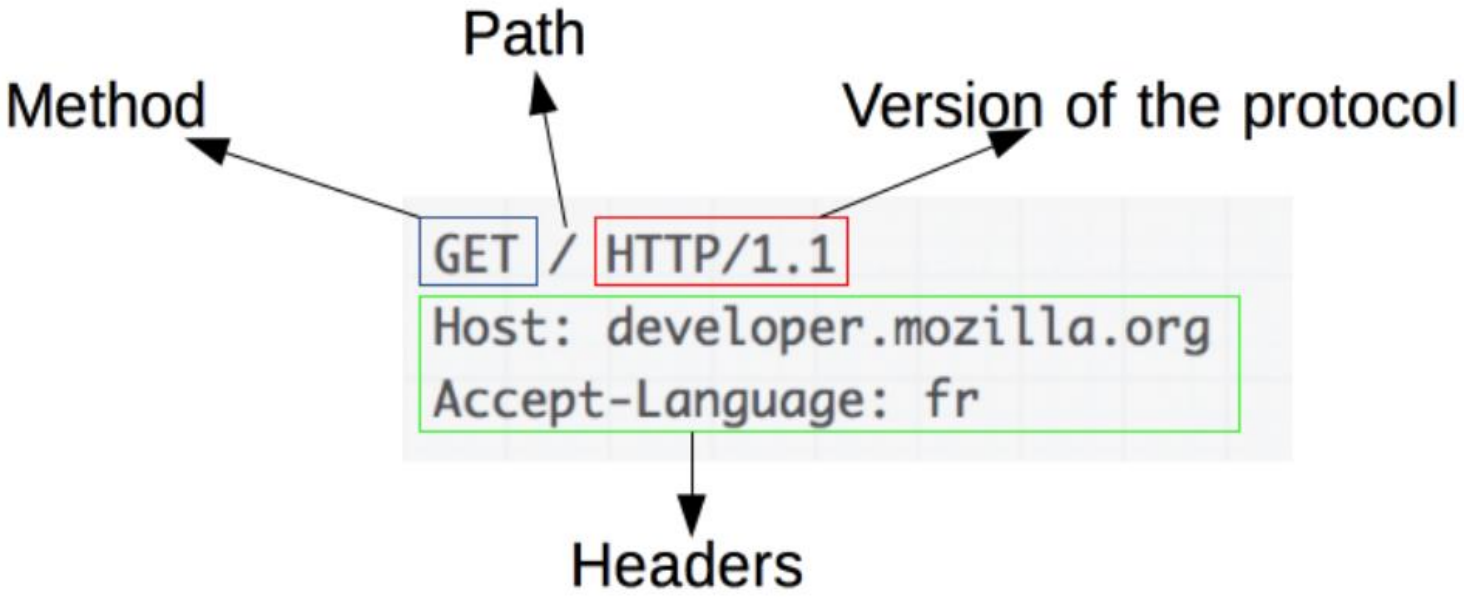
# Первые запросы

1. Jupyter hub:
   - Делаем запросы со стандартной urllib
   - Делаем запрос с помощью библиотеки requests

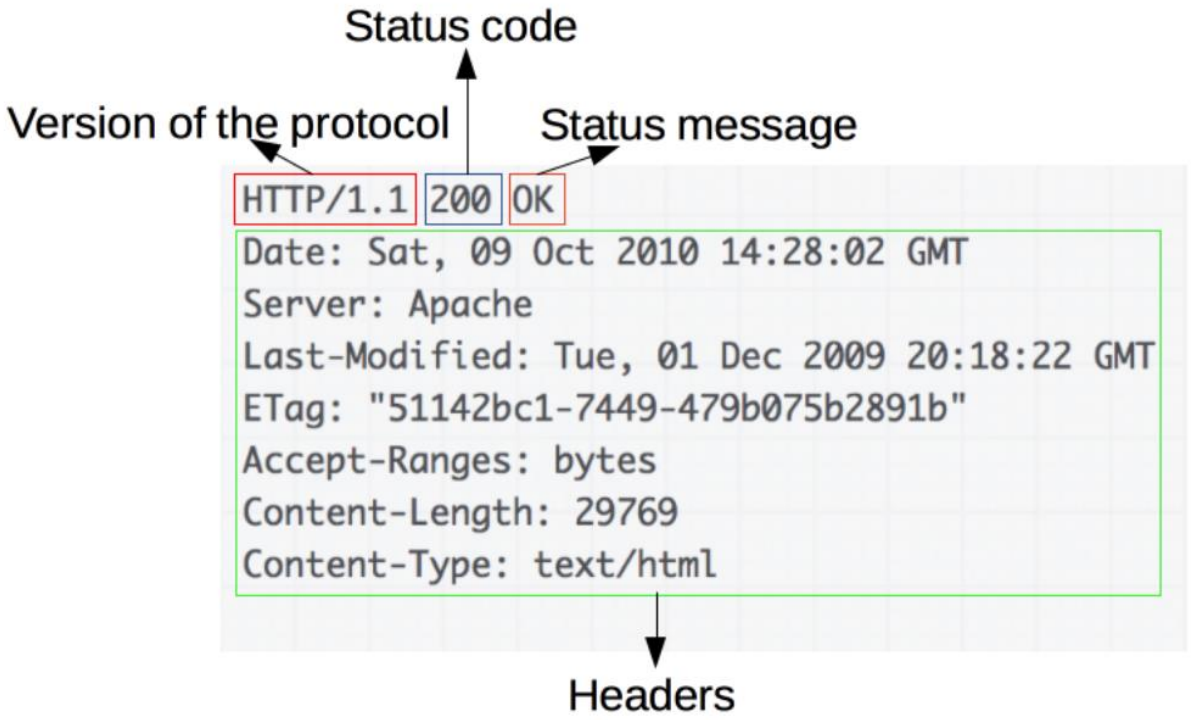
В requests имеется:
- Множество методов http аутентификации
- Сессии с куками
- Полноценная поддержка SSL
- Различные методы-плюшки вроде .json(), которые вернут данные в нужном формате
- Проксирование
- Грамотная и логичная работа с исключениями

# ВОПРОСЫ?
# И ПЕРЕРЫВ

# HTTP Запрос


Path
Method
Version of the protocol
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
Headers

# HTTP Ответ


Status code
Version of the protocol
Status message
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
Headers

# МЕТОДЫ И КОДЫ СОСТОЯНИЯ

Часто используемые методы:

- **GET** (запрос содержимого)
- **POST** (передача пользовательских данных)

Некоторые другие методы (используются например при реализации *RESTful API*):

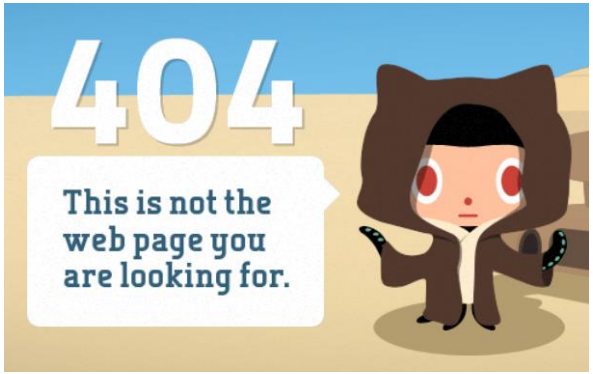
- **HEAD** (аналогично GET, но сервер должен вернуть только заголовки)
- **PUT**
- **PATCH**
- **DELETE**
- **OPTIONS**

Коды состояния:

- **2xx** – успешный запрос (сервер возвращает тело ответа)
- **3xx** – перенаправления (адрес перенаправления приходит в заголовке Location)
- **4xx** – клиентские ошибки
- **5xx** – серверные ошибки

# Наиболее используемые коды состояния

- 200 OK
- 201 Created
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not found
- 500 Internal Server Error
- 501 Not Implemented
- 502 Bad Gateway



# ЗАГОЛОВКИ

Примеры заголовков запроса:

- **Host** (доменное имя запрашиваемого ресурса)
- **User-Agent** (информация о браузере)
- **Referer** (информация о странице, с которой осуществляется переход)
- **Cookie** (значение Cookie, передаваемое обратно на сервер)

Примеры заголовков ответа:

- **Content-Length** (длина тела ответа)
- **Content-Type** (тип возвращаемых данных, например “text/plain”)
- **Location** (URL для перенаправления запроса на стороне клиента)
- **Set-Cookie** (значение Cookie для сохранения на стороне клиента)

# HTTP/2

**HTTP/2** (изначально **HTTP/2.0**) — вторая крупная версия протокола HTTP, появившаяся в 2015 году.

Разработан с целью увеличения производительности web-приложений, снижения нагрузки на сетевую инфраструктуру и облегчения написания высоконагруженных приложений.

Во многом повторяет концепции HTTP, но имеет ряд особенностей:

- бинарный протокол (вместо текстового)
- мультиплексирует нескольких запросов в одном TCP-соединении
- поддерживает технологию «server push»
- сжимает данные в заголовках
- ... и многое другое

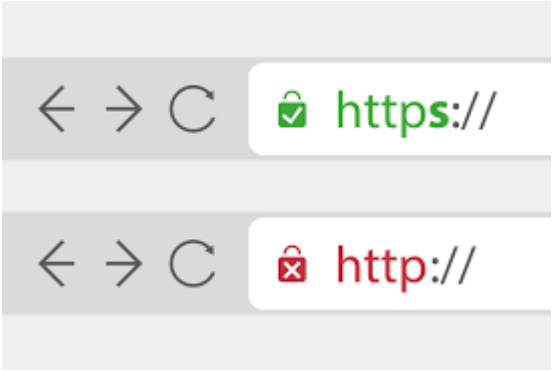
Большинство клиентских реализаций могут использовать HTTP/2 только поверх TLS (хотя формально по стандарту использование шифрования не обязательно).

# А что с безопасностью?

Для HTTP есть распространённое расширение, которое реализует упаковку передаваемых данных в криптографический протокол **SSL** или **TLS**.

Название этого расширения — **HTTPS** (*HyperText Transfer Protocol Secure*).

Для HTTPS-соединений обычно используется TCP-порт 443

Часть 3

# ПРАКТИКА

# Первое приложение на Flask

1. Создаём веб приложение с реализацией CRUD
2. Отправлять запросы через request

| HTTP Method | Action | Examples |
|---|---|---|
| GET | Obtain information about a resource | http://example.com/api/orders (retrieve order list) |
| GET | Obtain information about a resource | http://example.com/api/orders/123 (retrieve order #123) |
| POST | Create a new resource | http://example.com/api/orders (create a new order, from data provided with the request) |
| PUT | Update a resource | http://example.com/api/orders/123 (update order #123, from data provided with the request) |
| DELETE | Delete a resource | http://example.com/api/orders/123 (delete order #123) |

# Homework, задача “хорошо”

Взять любой python web frameworks на выбор и реализовать Web приложение с CRUD API:

| HTTP Method | Action | Examples |
|---|---|---|
| GET | Obtain information about a resource | http://example.com/api/orders (retrieve order list) |
| GET | Obtain information about a resource | http://example.com/api/orders/123 (retrieve order #123) |
| POST | Create a new resource | http://example.com/api/orders (create a new order, from data provided with the request) |
| PUT | Update a resource | http://example.com/api/orders/123 (update order #123, from data provided with the request) |
| DELETE | Delete a resource | http://example.com/api/orders/123 (delete order #123) |

Some ideas:
- Orders
- Books
- Recipes
- Todo list
- Exercise Tracker
- Journal
- Blog
- Banking App
- Text Editor
- Planning App
- Habit Tracker

# Homework, задача “отлично”

Взять любой python web frameworks на выбор, придумать и реализовать API (2-3 запроса) с интеграцией с внешними сервисами, т.е. будет делать запросы к внешним API из списка, например из Machine Learning, clarifai:

```
from clarifai.rest import ClarifaiApp

# Instantiate a new Clarifai app by passing in your API key.
app = ClarifaiApp(api_key='YOUR_API_KEY')

# Choose one of the public models.
model = app.public_models.general_model

# Predict the contents of an image by passing in a URL.
response = model.predict_by_url(url='https://samples.clarifai.com/metro-north.jpg')
```

# Материал

Архитектура веба, основы

Архитектура высоких нагрузок

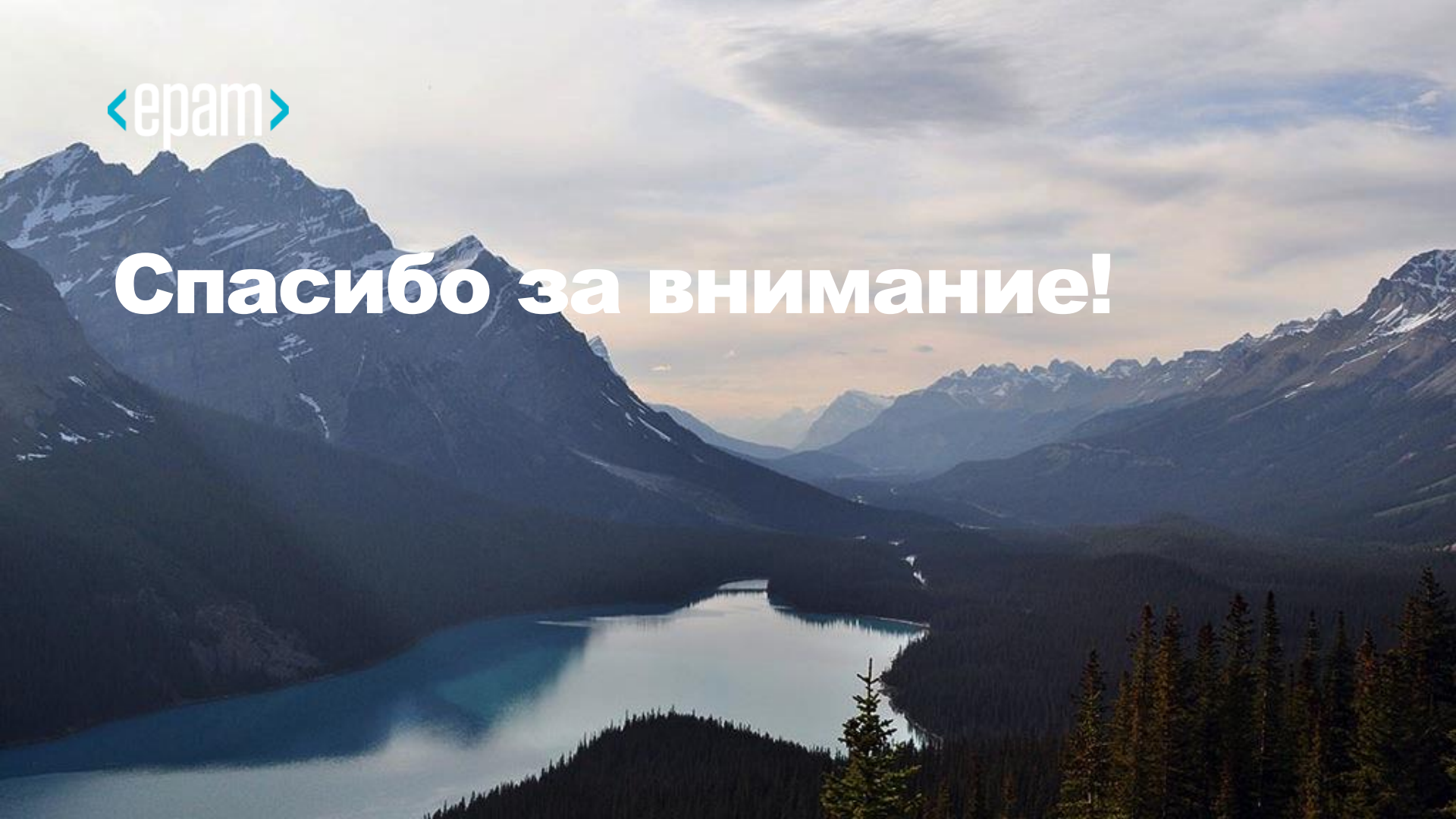
Введение в UWSGI серверы

Flask tutorial

List of web frameworks – python

List of public API

Apache vs NGINX


epam
Спасибо за внимание!