Урок 1. Что такое web server. Знакомство с серверами

Привет! За время курса мы с вами успели разобраться в достаточном количестве тем, чтобы суметь написать рабочее приложение, решающее какую-то реальную задачу. Одна проблема: мы не сможем развернуть наше приложение так, чтобы оно было в состоянии обслужить запросы от более-менее большого числа пользователей. Самые внимательные наверняка замечали, что когда мы запускаем Flask-приложение, то в консоли нам заботливо сообщают, что данный сервер нельзя использовать в production.

```
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead
```

Но что мы вообще знаем про веб-сервера, кроме того, что это некая программа и/или железо, которое обслуживает запросы от клиентов?
А как нам превратить нашу программу на Python, возможно, написанную на каком-то модном фреймворке вроде Flask, в сервер, способный обслужить больше 10 клиентов за раз? Давайте немного посмотрим на историю решения этой проблемы и разберёмся с стандартами/подходами, которые появились за долгое время существования World Wide Web.

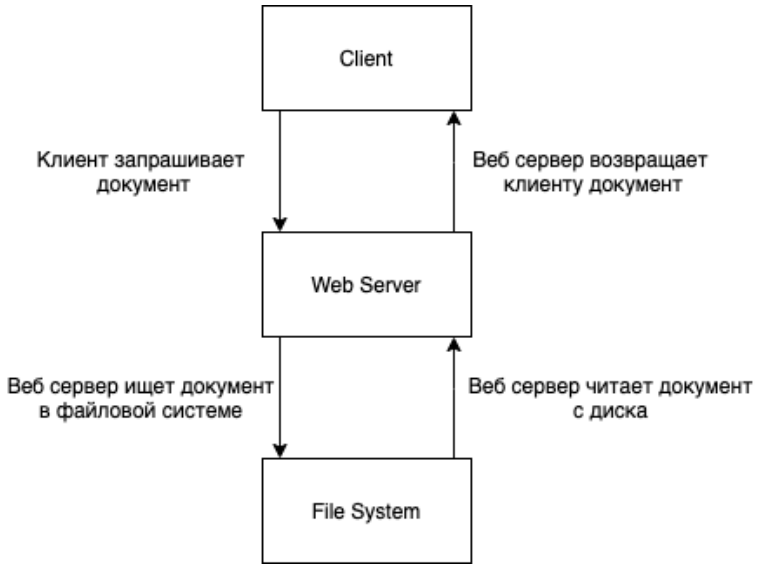
Начнём с самого верха. В какой момент начинает работать наш сервер? В момент запроса от клиента.

Клиент может запрашивать два типа страниц (мы ведь говорим о HTTP) — статическую и динамическую.

В случае запроса клиента на получение статической страницы, например по URL,

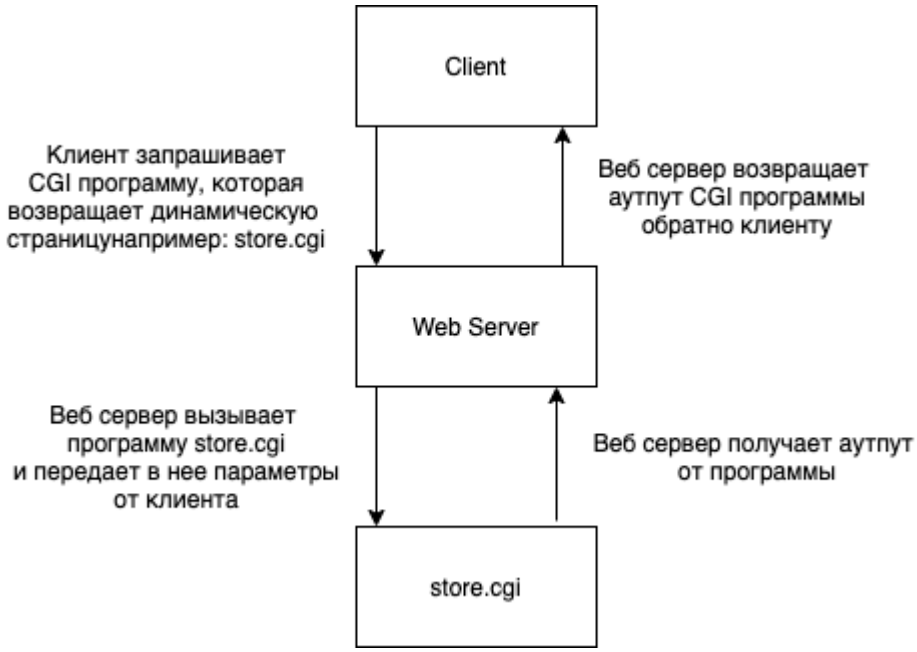
```
http://some-server.com/index.html
```

сервер по URL понимает, какой файл из файловой системы нужно отдать клиенту, и просто отдаёт его.
Этот нехитрый процесс изображён на диаграмме.



Пример такого сайта — типичный лендинг или сайт-визитка. Мы никак не будем взаимодействовать с пользователем, будем только отдавать ему контент уже подготовленной страницы. И, понятно, будь всё так просто, мы бы не учились с вами тут писать веб-приложения.
Но на заре интернета именно такой подход был тем, что стало основой World Wide Web: сервера хостили только статичные html-документы или картинки. Даже JS и CSS появились позднее.

Следующая важная веха в истории веб-серверов — появление концепта динамических страниц. Такая страница не просто дана юзеру, как книга и газета, а начинает взаимодействовать с ним напрямую или косвенно, реагировать по input от него или по метаинформации о клиенте и т. д. Наиболее старый и распространённый стандарт для осуществления этой магии — Common Gateway Interface. Из этого имени совершенно неясно, как эта магия должна случиться, но, по сути, он описывает то, как веб-сервера должны запускать программы локально для себя и отправлять output этих программ клиентам.



Обобщённый алгоритм работы через CGI можно представить в следующем виде:

1) клиент запрашивает CGI-приложение по его URL;
2) веб-сервер принимает запрос и устанавливает переменные окружения, через них приложению передаются данные и служебная информация;
3) веб-сервер перенаправляет запросы через стандартный поток ввода (stdin) на вход вызываемой программы;
4) CGI-приложение выполняет все необходимые операции и формирует результаты в виде HTML;
5) сформированный гипертекст возвращается веб-серверу через стандартный поток вывода (stdout); сообщения об ошибках передаются через stderr;
6) веб-сервер передаёт результаты запроса клиенту.

Удобство этого протокола в том, что серверу совершенно неважно, какую программу он будет вызывать. По сути, он точно так же, как и мы, работая в консоли, вызывает программу, передаёт в stdin данные, читает данные из stdout и передаёт их пользователю. Очевидно, что в таком подходе можно использовать любой язык программирования, главное, чтобы программа могла работать с потоками ввода-вывода. Интересный нюанс протокола: все параметры от пользователя передаются в программу как переменные окружения.

Плюсы данного подхода

- Процесс cgi-скрипта и процесс веб-сервера — это два разных процесса. И падение первого никак не скажется на состоянии второго.
- Можно использовать любой язык программирования для написания скрипта.

Минусы: физический сервер должен быть довольно производительным, ведь мы будем поднимать на каждый запрос от клиента новый процесс. А они иногда могут

подвисать. В итоге http-соединение с клиентом умрёт по timeout, а процесс скрипта продолжит жить. И на новый запрос это повторится опять.

Следующее эволюционное развитие протокола CGI — FastCGI (вообще интересно проследить за тем, как программисты любят называть новые подходы как Faster прежних). Если упрощать, то при написании этого протокола люди подумали: а что, если мы не будем запускать под каждый новый запрос отдельный процесс, а сделаем один процесс-демон (такой, который крутится где-то на бэкграунде и как будто невидим), который будет обслуживать запросы от пользователей один за другим? Задача веб-сервера теперь не просто вызвать нужную программу и передать ей на вход параметры, а передать данные в уже запущенный процесс. Это несколько сложнее, чем в случае с CGI, но выгода от сокращения времени на запуск отдельных программ огромна. Кстати, ничто не мешает запустить несколько таких процессов и балансировать нагрузку между ними. И мы можем запустить эти процессы на физически удалённых машинах — в общем, одни профиты.

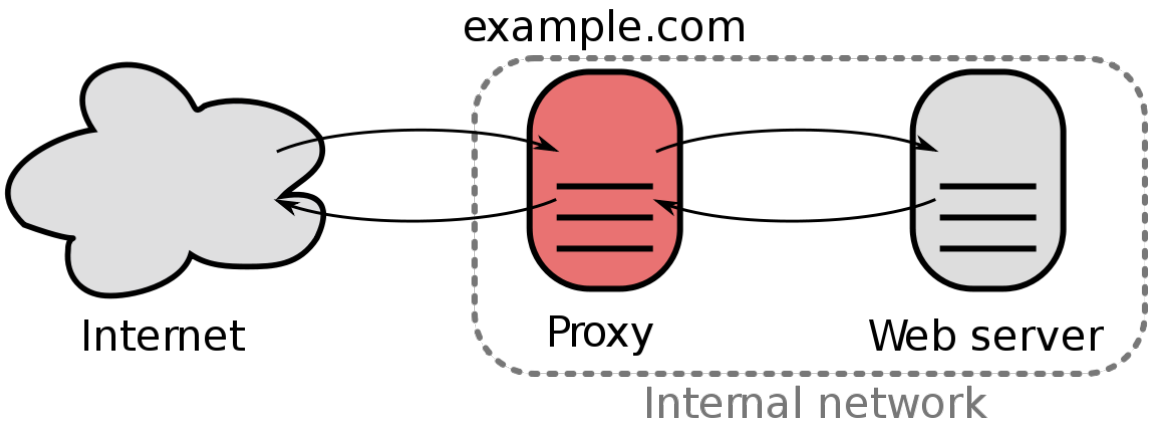
Теперь о том, какие веб-сервера (программы) бывают. На сегодняшний день есть два самых популярных сервера, способных как отдавать веб-статику, так и взаимодействовать с другими приложениями. Они вместе обслуживают более половины всего трафика в интернете. Это Apache и NGINX.

Пройдёмся кратко по их истории и основным отличиям.
Apache появился в 1995 году и сразу стал популярным продуктом «всё-в-одном» для хостинга в самом широком смысле. Поддержка огромного числа языков, куча настроек под любые нужды. Принцип работы такой: каждый запрос клиента создаёт новый процесс в системе. Какие минусы тут могут быть, мы уже знаем.

NGINX появился в 2004 году благодаря трудам нашего соотечественника Игоря Сысоева как решение проблемы C10K — обслужить 10 000 одновременных запросов. Достигается это за счёт асинхронной архитектуры. В дополнение к этому легковесность, масштабируемость и переносимость позволяют использовать NGINX почти на любом железе.

Специализация NGINX — хостинг статики. В этом он более чем в два раза быстрее Apache. А вот с динамическими страницами сложнее: сам NGINX не умеет генерировать динамический контент из коробки. Вместо этого он проксирует запрос к тем, кто это умеет делать. Например, в тот же Apache. В Python-стеке это NGINX+uWSGI/Gunicorn. Подробнее про них мы поговорим чуть позднее. В данном типе работы NGINX и Apache равны по скорости.

Кстати, такой подход, когда мы обращаемся к серверу, а он прокидывает запрос в другое место, скрытое от пользователя, называется reversed proxy. Сервер служит точкой входа во внутреннюю сеть и транслирует запросы от клиента на нужные ресурсы.



Кстати, по этому пути (статика — cgi — fastcgi) интересно наблюдать за тем, как развивался интернет. Что-то вроде того, как наблюдать за эмбрионами и их превращениями в такт эволюции.

Итак, теперь мы ещё лучше понимаем, кто это такой — веб-сервер, какие задачи он решает и какие программы серверов бывают. На следующем уроке мы с вами на практике познакомимся с NGINX и научимся отдавать статику.