Лабораторная работа №5

Настройка http- и веб-сервера

В. Слюсарь

Ноябрь 2022

Цель работы

На примере python и nginx освоить основые принципы современной веб-инфраструктуры.

Порядок выполнения работы

1. Изучите теоретические сведения.
2. Согласно инструкции, запустите http сервер, используя встроенный python модуль.
3. Согласно инструкции, запустите веб-сервер nginx.
4. Скопируйте форму отчета о проделанной работе, приведенную в конце данного документа, в текстовый или табличный редактор. Сделайте скриншоты вашей веб-страницы, заполните отчет. Результаты сохраните на личный диск.
5. Ответьте на контрольные вопросы.
6. Предъявите готовый отчет преподавателю.

Теоретические сведения

Протокол HTTP

HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») — текстовый протокол прикладного уровня. На данный момент используется повсеместно и стал де-факто стандартом обмена практически любой информацией между различными сервисами в сети интернет.

В основе протокола лежит модель «клиент-сервер» («запрос-ответ»), которая подразумевает, что существует некоторый **клиент**, желающий получить или отправить ту или иную информацию. Клиент отправляет **запрос серверу**, который обратывает этот запрос тем или иным образом, и возвращает клиенту **ответ**.

К примеру, предположим что клиент хочет просмотреть какую-то веб-страницу. Тогда клиент отправит серверу так называемый GET запрос, в котором укажет имя запрашиваемого файла. Сервер, в свою очередь, загрузит этот файл с диска и отправит его содержимое клиенту в качестве ответа на запрос.

HTTP сервер

Программу, умеющую обрабатывать HTTP запросы и формировать HTTP ответы, называют HTTP сервером. HTTP сервер обычно умеет много больше, чем просто отдавать файлы в неизменном виде с диска. Современные HTTP серверы часто имеют код для обращения к различным базам данных, умеют наполнять шаблоны страниц, а также в свою очередь делать HTTP запросы к другим серверам.

Веб-сервер

Строго говоря, веб-серверами называют как раз HTTP/HTTPS сервера. Однако, на сегодняшний день этот термин часто используется для описания специальных программ, которые работают *вместе* с HTTP серверами. В обязанности веб-сервера часто входит: шифрование/дешифрование HTTPS трафика, балансировка нагрузки, обратное проксирование и тд. Наиболее популярные на сегодняшний день веб-серверы: Nginx, Apache, OpenResty, Cloudflare Server, GWS, IIS, Caddy и др.

Все перечисленные веб-серверы поддерживают весьма устрашающее количество функций, из которых мы остановимся только на нескольких. Первая из них — **обратное проксирование** (англ. reverse proxy). При создании веб-инфраструктуры нередко оставляют только один сервер с доступным из интернета IP-адресом. Это делается, в том числе, из соображений безопасности: остальные сервера, в том числе базы данных, имеет смысл вообще не делать доступными из интернета. Однако, если клиенты могут делать запросы только к одному серверу, то как нам, к примеру, запустить два разных сайта? Это одна из ситуаций, в которой может быть использовано обратное проксирование. На сервер, доступный из интернета, устанавливается веб-сервер, который принимает все запросы от клиентов и, в зависимости от содержимого запроса, *перенаправляет* запрос какому-нибудь HTTP серверу во внутренней сети. Получив ответ от HTTP сервера, веб-сервер просто перенаправляет ответ клиенту.

Вторая интересующая нас функция — **разграничение доступа**. Веб-серверы подразумевают возможность использования разных сетевых интерфейсов. К примеру, если указать

при настройке веб-сервера IP адрес 127.0.0.1, то он будет принимать HTTP запросы только с того же самого компьютера, на котором запущен сам. Если мы используем внутренний IP адрес (что-нибудь вроде 192.168.10.62), то веб-сервер уже будет принимать запросы от других компьютеров внутри нашей сети. Если же мы используем внешний IP адрес (если он у нас вообще есть), то веб-сервер будет принимать запросы из всего интернета.

Последняя важная для нас особенность веб-серверов — они обычно работают на портах 80/443 (стандартные порты для HTTP/HTTPS). При вводе url в адресную строку браузера вы обычно пишете только сам адрес, не указывая порт. На самом деле этот порт добавляется автоматически: 80 для HTTP и 443 для HTTPS. Но тогда мы можем иметь максимум один HTTP и один HTTPS сервер на каждом компьютере (ведь на одном порту может работать только одна программа)? Эту проблему также за нас решает веб-сервер. Его обычно запускают на **стандартном порту 80 или 443**, после чего веб-сервер уже перенаправляет запросы HTTP серверам, запущенным на других портах.

В практической части этой лабораторной работы мы запустим два HTTP сервера на разных портах и один веб-сервер, который будет перенаправлять запросы на нужный HTTP сервер в зависимости от url.

Практическая часть: запуск http серверов на python

Чтобы не потерять суть за деталями, мы будем использвать уже готовый HTTP сервер. Такой присутствует, в том числе, в стандартном наборе модулей python.

1. Создайте на рабочем столе две папки с названиями netlab_site1 и netlab_site2. Если такие папки уже есть, то удалите их содержимое.

2. В каждой из этих папок создайте файл index.html с произвольным содержимым. Это будут ваши веб-страницы. Обратите внимение, что это должны быть файлы с расширением .html, а не файлы index.html.txt. Если вы будете создавать их в блокноте, то обязательно выбирайте в выпадающем списке «Тип файла» пункт «Все файлы».

3. Откройте терминал, и с помощью утилиты cd перейдите в папку python-3.11.2, приложенную к лабораторной работе.

4. Запустите первый http сервер, введя команду

   ```
   python.exe -m http.server 8000 --bind 127.0.0.1 --directory
   C:\path\to\netlabsite1
   ```

5. Откройте еще один терминал, и по аналогии с первым запустите второй http сервер, введя команду

```
python.exe -m http.server 8001 --bind 127.0.0.1 --directory
C:\path\to\netlabsite2
```

Если вы всё сделали правильно, то вы должны увидеть в терминале текст, похожий на Рис.1.

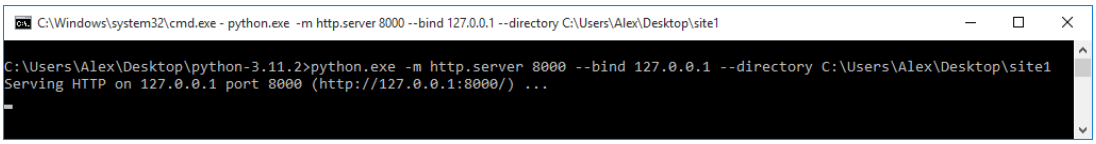

Рис. 1: HTTP сервер запустился

Помимо этого, вы можете открыть веб-браузер и, введя в адресную строку 127.0.0.1:8000, увидеть содержимое файла index.html из папки netlabsite1, а введя 127.0.0.1:8001 — содержимое index.html из папки netlabsite2 (см. Рис.2).

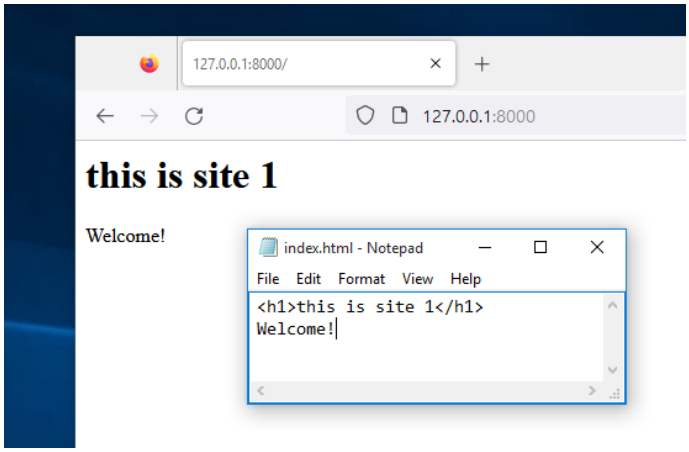

Рис. 2: Веб-страница отобразилась

Практическая часть: запуск веб-сервера nginx

Мы будем использовать самый популярный на сегодняшний день веб-сервер: Nginx.

1. Откройте папку nginx-1.22.1 и запустите двойным кликом файл nginx.exe. Если до этого nginx на этом компьютере не запускали, то он попросит разрешения для создания исключений в файерволе. Ему нужно разрешить сделать то что он просит, это необходимо для работы.

2. Зайдите в папку conf, и создайте внутри нее папку conf.d. Если папка conf.d уже есть, то удалите ее содержимое.

3. В папке conf.d создайте файл с произвольным именем и расширением .conf, например labsite.conf. Внутри этого файла мы настроим обратное проксирование nginx до наших двух http серверов.

4. Откройте созданный файл, и впишите в него настройки по аналогии с приведенными:

```
server {
    listen 127.0.0.1;
    location /first/ {
        proxy_pass http://127.0.0.1:8000/;
    }
    location /second/ {
        proxy_pass http://127.0.0.1:8001/;
    }
}
```

   При такой настройке все запросы `http://127.0.0.1/first` будут передаваться на http сервер на порту 8000, а запросы `http://127.0.0.1/second` будут передаваться на http сервер на порту 8001. Замените пути first и second на какие-то другие.

5. Чтобы изменения конфигурационного файла nginx применились, нужно перейти из терминала в папку nginx-1.22.1 и выполнить команду `nginx -s reload`.

6. Если вы всё сделали правильно, то при вводе в браузере адреса 127.0.0.1/first вам отобразится веб-страница из папки netlabsite1, а при вводе адреса 127.0.0.1/second — страница из папки netsitelab2 (см. Рис.3).

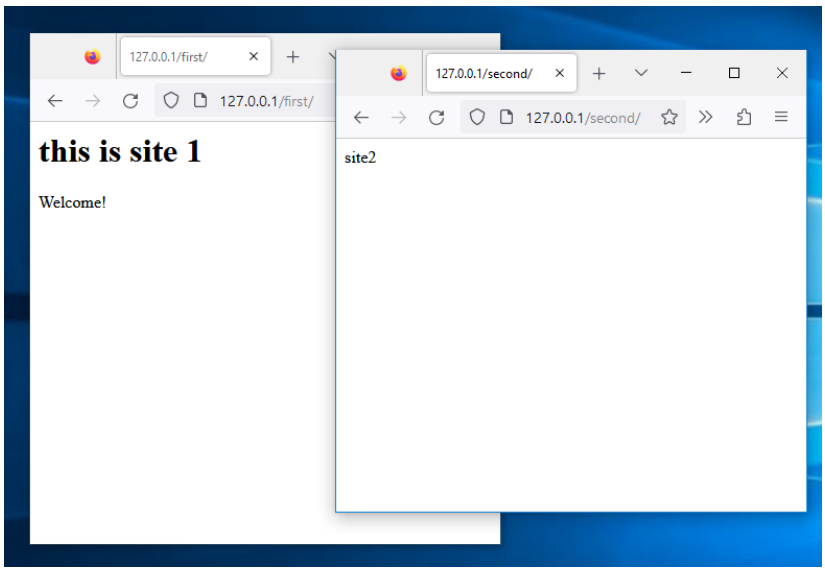

Рис. 3: Веб-страницы, полученные от веб-сервера

7. Остановите HTTP сервера, закрыв их терминалы или нажав Ctrl+C.
8. Остановите nginx через диспетчер задач, либо из консоли командой `nginx -s stop`.

Контрольные вопросы

1. На каком (по номеру) уровне модели OSI находится протокол HTTP?
2. Можно ли как-то избежать того, чтобы HTTP сервер каждый раз при запросе файла читал его с диска?
3. Как вам кажется, какие еще функции кроме перечисленных (обратное проксирование, разграничение доступа, проксирование портов) может выполнять веб-сервер?

Отчет к лабораторной работе №5

Задание №1

Запустите два HTTP сервера на python и веб-сервер Nginx согласно приведенной инструкции. Прикрепите к отчету ваш конфигурационный файл nginx из папки conf.d, скришот, на котором видно что по разным адресам отображаются страницы, отданные разными http серверами, а также ответы на вопросы:

- Какие пути (url) вы использовали для проксирования каждого сервера?
- Как вы назвали конфигурационный файл nginx?
- Каким образом вы поняли, что запросы доходят до http серверов?

Задание №2*

Попробуйте перенастроить nginx таким образом, чтобы ваши странички могли посмотреть ваши соседи со своих компьютеров. Для этого вам нужно будет изменить директиву server_name в конфигурационном файле. Не забудьте сделать `nginx -s reload`.

Попробуйте сделать так, чтобы при заходе по адресу 127.0.0.1 вам отображалась одна страница, а при заходе по внутреннему IP — другая. Вам нужно будет создать несколько директив server в конфигурационном файле nginx.