Лекция 6. Веб-сервер на Linux

Цель лекции

- Понять основы веб-приложений
- Изучить принципы работы веб-сервера
- Разобрать схему обратного прокси-сервера
- Научиться настраивать стандартный стек для веба
- Познакомиться с СУБД MySQL

Термины

HTTP (HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.
HTTPS — расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. Данные в протоколе HTTPS передаются поверх криптографических протоколов TLS или устаревшего в 2015 году SSL. В отличие от HTTP с TCP-портом 80, для HTTPS по умолчанию используется TCP-порт 443.
TLS (transport layer security — Протокол защиты транспортного уровня), как и его предшественник SSL (англ. secure sockets layer — слой защищённых сокетов), — криптографические протоколы, обеспечивающие защищённую передачу данных между узлами в сети Интернет. TLS и SSL используют асимметричное шифрование для аутентификации, симметричное шифрование для конфиденциальности и коды аутентичности сообщений для сохранения целостности сообщений.
Удостоверяющий центр (CA – certificate authority) — доверенная организация, которая имеет право выпускать сертификаты юридическим и физическим лицам.
СУБД – система управления базами данных, комплекс ПО для работы с базами данных определённого типа (например, реляционными).

План

1. Компоненты веб-приложения.
2. Взаимодействие браузера и веб-сервера.
3. Основы HTTP-протокола.
4. Работа HTTPS и TLS (SSL). Сертификаты и удостоверяющие центры.
5. Обзор возможностей веб-серверов Nginx.
6. Обзор веб-сервера Apache.
7. Динамический и статический контент, схема Reverse Proxy.
8. Установка и запуск СУБД MySQL.
9. Установка PHP, настройка модуля Apache.
10. Установка и запуск демона PHP-FPM.

Текст лекции

Компоненты веб-приложения

Веб-приложения мы используем каждый день. Многие системы перешли на веб-интерфейс из-за широких возможностей, доступности и совместимости с различными устройствами.
Большинство веб-приложений использует Linux как платформу для серверной части. Здесь самое время познакомиться с двумя компонентами: frontend (клиентская часть) и backend (серверная часть). Клиентская часть работает в браузере и обычно решает задачи отображения и взаимодействия с интерфейсом. Серверная часть отвечает за работу бизнес-логики, обмен данными и вычисления. Сразу уточним, что термины backend и frontend универсальны и могут применяться в других контекстах, например в серверной части это могут быть два сервера, относящихся к разным частям архитектуры. При этом frontend это всегда часть, которая ближе к пользователю, а backend находится дальше него.
Мы будем рассматривать именно серверную часть веб-приложений, потому что именно она работает на базе нашей системы Linux.
Посмотрим, как клиентская и серверная части связаны друг с другом.

Взаимодействие браузера и веб-сервера

Браузер в веб-приложении выполняет роль клиента. Он должен отправлять запросы и получать ответы от веб-сервера. Для взаимодействия между браузером и сервером используется протокол HTTP, который мы рассмотрим позже.
Для начала работы с веб-приложением (сайтом) браузер должен получить IP-адрес сервера, то есть преобразовать доменное имя (хост) в IP через систему DNS. Далее происходит подключение по порту `80` (`HTTP`) или `443` (`HTTPS`). Обычно используется протокол `TCP`, но в новой версии протокола `HTTP/3` задействован `UDP`.
Заменить браузер для тестирования в консоли можно утилитами `wget` и `curl`.
Утилита `wget` предназначена для загрузки файлов по HTTP или HTTPS. Например:

```
wget https://yastatic.net/jquery/2.1.4/jquery.min.js
```

Утилита curl по умолчанию показывает содержимое документа, который мы скачиваем.

```
curl -L https://ya.ru/
```

Опция `-L` включает режим поддержки редиректов (перенаправлений) клиента.
Подробнее взаимодействие браузера и веб-сервера раскрывается в механизмах протокола HTTP, которые мы рассмотрим ниже.

Основы HTTP-протокола

HTTP – это основной протокол работы любого веб-приложения. Сегодня он используется с различными надстройками и модификациями (HTTP/2, HTTP/3, HTTPS), но суть остаётся неизменной.
В основе взаимодействия лежат запрос (request) и ответ (response). Запросы отправляет клиент в сторону сервера, который отвечает на эти запросы.

В запросе можно разделить несколько частей: строка запроса с методом, заголовки и тело запроса. Метод является режимом запроса (GET, POST, PUT, DELETE и т.д.), к методу нужно добавить URL (адрес).
Заголовки запроса уточняют его, дают дополнительные данные о клиенте (поддерживаемые технологии, ОС, версию клиента и т.д.)
В некоторых запросах присутствует тело запроса. Обычно это запросы на загрузку данных на сервер (POST). В большинстве запросов тело запроса пустое.
Ответ также содержит строку статуса, в которой есть код ответа (200, 302, 404, 500 и т.д.), текстовое пояснение и версия протокола. Коды ответа делятся по типам, которые связаны с первой цифрой кода: 2xx – нормальный ответ, 3xx — перенаправление, 4xx – ошибка клиента, 5xx — ошибка сервера.
Далее указываются заголовки ответа, которые описывают тип данных (текст, картинка, бинарный файл), дату изменения, возможности кэширования на клиенте, размер и другие подробности.
После заголовков следует тело ответа. В большинстве ответов тело имеет ненулевой размер и содержит документ. Ответы без тела также бывают, но значительно реже (обычно редиректы или ответы на запросы с методом `HEAD`).
Ранние версии протокола HTTP разрабатывались без учета требований безопасности, поэтому всё взаимодействие между браузером и сервером происходит в виде открытых текстовых сообщений. То есть, в случае использования общественных сетей существует риск перехвата и искажения сообщений. Для решения этой проблемы были разработаны расширения протокола HTTP — SSL и позже TLS, которые вместе с HTTP образуют защищенную версию: HTTPS.

Работа HTTPS и TLS (SSL). Сертификаты и удостоверяющие центры

В современном мире большинство веб-приложений работает по защищенному протоколу TLS, который является транспортным дополнением, обеспечивающим безопасный обмен данными. Все трафик при использовании TLS передаётся в зашифрованном виде, ключ шифрования устанавливается индивидуально для каждой сессии.
TLS является развитием протокола SSL, который сегодня практически не используется из-за множественных уязвимостей. Актуальные (рекомендуемые к использованию) версии TLS на сегодня это `1.2` и `1.3`.
Использование SSL/TLS с HTTP обычно обозначается как HTTPS. Защита соединения состоит не только в шифровании данных, но также в подтверждении имени домена. Когда мы подключаемся к сайту, веб-сервер предоставляет сертификат, в котором отмечены доменные имена, дата действия и указание на удостоверяющий центр, который выдал этот сертификат. Браузер принимает сертификат, проверяет его легитимность по списку доверенных корневых сертификатов в операционной системе. Если сертификат выдан одним из доверенных удостоверяющих центров (CA), то браузер проверяет соответствие доменного имени сайта и доменов в сертификате. Если сертификат доверенный, не просрочен, доменное имя соответствует, то браузер разрешит подключение к этому серверу.

То есть, TLS позволяет идентифицировать сайт по доменному имени, получить подтверждение о том, что владелец сервера действительно владеет доменом и защитить обмен данными стойким шифрованием.
Требование работать по защищённому протоколу может вызывать неудобства при разработке в локальной среде. В этом случае имеет смысл использовать рабочий сертификат для домена и направить этот домен на локальный адрес (`127.0.0.1`).
Перейдём к обзору популярного веб-сервера Nginx.

Веб-сервер Nginx

Сегодня Nginx это самый популярный веб-сервер в мире. Первоначально он создавался как легковесная альтернатива веб-серверу Apache, который долго удерживал первенство в мире веб-серверов. Nginx может очень эффективно обрабатывать большое количество (десятки и сотни тысяч) соединений и обладает широкой функциональностью. Построен на базе модульной архитектуры, что позволяет расширять его возможности.
Существуют две версии Nginx – открытая и коммерческая (Nginx Plus). Мы будем рассматривать открытую версию.
Для конфигурации используются файлы в директории `/etc/nginx`. С точки зрения синтаксиса в конфигурации есть блоки (части, объединённые фигурными скобками `{}`) и директивы (однострочные или многострочные выражения). Директивы всегда заканчиваются точкой с запятой (`;`). Все директивы конфигурации имеют свой контекст применения и подчиняются общему принципу: действует наиболее конкретное значение директивы (на самом глубоком уровне), если значение не указано, то применяется значение по умолчанию.
Конфигурация Nginx подробно описана на сайте продукта (http://nginx.org/ru/docs/).
Установка Nginx:

```
sudo apt install nginx
```

Настройки сайтов располагаются в рамках блока `server {}`.
Пример конфигурации сайта по умолчанию.

```
server {
	listen 80 default_server;
	listen [::]:80 default_server;

	root /var/www/html;

	index index.html index.htm index.nginx-debian.html;

	server_name _;

	location / {
		try_files $uri $uri/ =404;
	}
}
```

В этой конфигурации описывается сайт, который будет работать по умолчанию, на 80 порту (HTTP). Корневая директория сайта расположена по адресу `/var/www/html`.
После внесения изменений в конфигурацию полезно проверить синтаксис следующей командой:

```
sudo nginx -t
```

Если всё в порядке, применить конфигурацию:

```
sudo systemctl reload nginx
```

Эта команда применяет конфигурацию без разрыва соединений и потери запросов от пользователей.

Веб-сервер Apache

Apache долгое время был самым распространённым веб-сервером в Интернете, но недавно уступил серверу Nginx. Apache имеет развитую функциональность для решения любых задач веб-сервера. Его возможности определяются большим количеством стандартных модулей.
С точки зрения эффективности использования ресурсов он уступает Nginx, но пригоден для большинства применений за счет современных мультипроцессорных модулей (MPM).
Конфигурация Apache расположена в каталоге `/etc/apache2`. Принцип работы директив такой же, как и в Nginx. Синтаксис различается. Например, директивы разделяются новой строкой, блоки имеют другой синтаксис, например:

```
<Directory />
      Options FollowSymLinks
      AllowOverride None
      Require all denied
</Directory>
```

При одновременной работе Nginx и Apache необходимо выделить разные порты для их работы. По умолчанию веб-сервера слушают `80` порт, поэтому можно в конфигурации портов Apache (`/etc/apache2/ports.conf`) поставить другой номер порта, например `8080 (Listen 8080)`. Также можно ограничить доступ к Apache только локальным интерфейсом: `Listen 127.0.0.1:8080`.
После внесения изменений в конфигурацию не забываем проверить синтаксис:

```
sudo apachectl -t
```

И применить новую конфигурацию:

```
sudo systemctl reload apache2
```

Теперь пора посмотреть на совместную работу веб-серверов Nginx и Apache по схеме обратного прокси (reverse proxy).

Динамический и статический контент. Reverse proxy

В любом веб-приложении можно выделить два типа запросов (контента): динамические и статические.
Динамический контент это результат работы программного кода на серверной стороне. Например, часто HTML-код страниц является как раз динамическим контентом: для его

получения на сервере выполняется программный код (например, PHP), из которого отправляются запросы к базам данных и в результате формируется ответ.
Статический контент это обычные файлы, которые расположены на диске веб-сервера. При запросе такого контента веб-серверу нужно всего лишь прочитать файл и отправить его в ответе. Примеры статических запросов: картинки, файлы листов стилей (CSS), скрипты JavaScript (JS), шрифты. Логика обработки статических запросов намного проще.
Именно это разделение по типам запросов лежит в основе схемы обратного прокси. В этой схеме Nginx выступает в виде прокси-сервера, который обрабатывает статические запросы, а динамические отправляет на другой сервер (например, PHP-FPM или Apache). В этой схеме Nginx иногда называют frontend сервер, а Apache – backend сервер.
Таким образом, мы получаем максимальную эффективность веб-сервера Nginx и можем использовать любые языки программирования и связанные с ними веб-сервера. Например, Nginx не может напрямую выполнять PHP-код, поэтому динамические PHP-запросы нужно отправить на другой сервер. При этом с браузером в любых случаях будет взаимодействовать Nginx с поддержкой всех необходимых технологий.
Для настройки Nginx в качестве обратного прокси нужно настроить несколько блоков location. Один будет перехватывать статические запросы, а другой — переадресовывать запросы на Apache.
Пример такой конфигурации:

```
location / {
      proxy_pass http://localhost:8080;
      proxy_set_header Host $host;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Real-IP $remote_addr;
}

location ~* ^.+.(jpg|jpeg|gif|png|ico|css|zip|pdf|txt|tar|js)$ {
     root /var/www/html;
}
```

Нижний блок будет иметь приоритет, так как это регулярное выражение. За счет этого блока мы обрабатываем статические запросы на основе расширений файлов.
Верхний блок будет передавать запросы на Apache. Сюда попадут все запросы, которые не попали в нижний блок.
Итак, мы научились настраивать схему обратного прокси, теперь запустим динамический контент на примере PHP-модуля в Apache.

Установка PHP, настройка модуля Apache

Для обработки PHP через Apache нам понадобится сам интерпретатор php и пакет модуля Apache, который свяжет веб-сервер с интерпретатором.
Чтобы установить всё сразу можно использовать команду:

```
apt install libapache2-mod-php8.1 php8.1
```

Эта команда установит сразу базовый пакет с PHP версии 8.1 и соответствующий модуль для Apache mod_php. Важно заметить, что в PHP существуют свои модули (расширения) которые могут устанавливаться отдельно, например:

```
apt install php8.1-zip php8.1-imagick
```

Сразу после установки веб-сервер Apache будет обрабатывать PHP-файлы с помощью модуля mod_php, который мы установили. Проверить это легко: создадим файл info.php в корне основного сайта для Apache (по умолчанию это `/var/www/html`) со следующим содержанием:

```
<?php
phpinfo();
?>
```

Далее можно зайти браузером в виртуальной машине по адресу: http://localhost/info.php.

Если всё работает, мы должны увидеть длинную страницу со статусом PHP, на которой видно, что запрос обработан через Apache.

Альтернативный вариант – запуск PHP-кода через отдельный сервер PHP-FPM, который может заменить Apache в нашей схеме обратного прокси.

Установка и запуск демона PHP-FPM

Установить сервер PHP-FPM можно командой:

```
apt install php8.1-fpm
```

В списке процессов появятся несколько процессов php-fpm. Конфигурация этого сервиса находится по адресу: `/etc/php/8.1/fpm/.`

Для того, чтобы использовать этот метод исполнения PHP нужно внести следующую секцию в конфигурацию сайта в Nginx.

```
location ~ \.php$ {
            include snippets/fastcgi-php.conf;
            root /var/www/html;
            fastcgi_pass unix:/run/php/php8.1-fpm.sock;
}
```

Этот блок будет получать все запросы к файлам *.php и отправлять их на php-fpm, который по умолчанию слушает UNIX-сокет по адресу: `/run/php/php8.1-fpm.sock.`

Для проверки работы нужно зайти на тот же адрес (http://localhost/info.php) и найти упоминание сервиса PHP-FPM в выводе скрипта в браузере.

Мы научились запускать PHP-код с применением нескольких серверов, теперь нужно узнать, как можно установить и запустить базу данных с использованием MySQL.

Установка и запуск СУБД MySQL

Система управления базами данных MySQL лидирует по использованию в веб-приложениях благодаря своей простоте и широким возможностям. На сегодня актуальная версия MySQL имеет обозначение 8.0. Мы будем использовать официальную версию от компании Oracle.

Установка сервера MySQL выполняется одной командой:

```
apt install mysql-server-8.0
```

Кроме сервера автоматически будет установлен клиент для работы с MySQL. Команда для запуска клиента: `mysql`.
Однако, нужно учитывать, что в MySQL есть своя система авторизации (пользователи и права доступа), поэтому для входа в администратора MySQL (пользователь root, не путать с пользователем root ОС), необходимо зайти через sudo:

```
sudo mysql
```

Эта команда даёт полный доступ к управлению базами данных и пользователями. Работает эта команда на основе связи системного пользователя root и внутреннего пользователя root для MySQL.
В консоли mysql мы можем выполнить несколько базовых команд, чтобы удостовериться в корректной работе сервиса.
Посмотрим список баз данных:
show databases;
Зайдём в системную БД mysql и посмотрим список пользователей.

```
use mysql;
SELECT * FROM user\G
```

Создадим новую базу данных и таблицу в ней:

```
CREATE DATABASE gb;
CREATE TABLE test(i INT);
```

Создадим несколько записей в новой таблице и проверим, что они появились:

```
INSERT INTO test (i) VALUES (1),(2),(3),(4);
SELECT * FROM test;
```

Отлично, MySQL работает и мы готовы запускать веб-приложения с этой СУБД.

Итоги занятия

- Разобрали, из чего состоит веб-приложение.
- Изучили основы протокола HTTP.
- Узнали, что такое HTTPS.
- Запустили веб-сервер Nginx.
- Настроили схему обратного прокси с использованием Nginx.
- Разобрали деление контента на динамический и статический.
- Познакомились с сервисом PHP-FPM.
- Установили СУБД MySQL и проверили базовые команды.