Основы развертывания приложений

Установка и настройка SSH-сервера, WEB-сервера и сайтов

Установка и настройка SSH-сервера (openssh-server)

Введение

- Рано или поздно наступит тот момент, когда нам потребуется развернуть наше приложение на сервере, посредством которого наши услуги будут поставляться конечному потребителю. Сам сервер мы можем добыть разными способами. Во-первых, можно арендовать т.н. VPS (Virtual Private Server) у какой-либо компании, которая занимается интернет-услугами и предоставлением хостинга. Во-вторых, есть возможность воспользоваться облачными сервисами (например, AWS), которые также предоставляют сервера и всю необходимую инфраструктуру. Наконец, в-третьих, можно попытаться развернуть полностью свой сервер и поддерживать его работоспособность вручную. Так или иначе, у нас будет некий удаленный компьютер, подключенный к сети Интернет.

- Чтобы взаимодействовать с этим компьютером (устанавливать зависимости, запускать и обновлять наши приложения), нам необходимо время от времени к нему подключаться. Обычно это осуществляется при помощи SSH (Secure Shell) протокола - изначально мы устанавливаем на компьютере SSH-сервер, который позволяет производить удаленные подключения к компьютеру и после процедуры аутентификации заходить в оболочку командной строки с правами какого-либо пользователя.

Установка SSH-сервера

- Предположим, что внутри нашего сервера используется операционная система Ubuntu (или какая-либо другая система, которая является производной от Debian). Перед установкой SSH-сервера нам необходимо любым способом попасть на наш сервер и начала установить и запустить firewall, чтобы уберечься от возможных проблем с безопасностью:

```
# установка firewall
sudo apt install -y ufw

# запуск firewall
sudo ufw enable
```

- Теперь пришла пора заняться непосредственно SSH-сервером - для этого надо выполнить следующую команду, которая установит пакет **openssh-server**:

```
sudo apt install -y openssh-server
```

- Далее запустим SSH-сервер нижеприведенной командой:

```
sudo systemctl start ssh
```

Настройка SSH-сервера

- Настала пора настроить наш SSH сервер для правильной и безопасной работы - для этого мы должны открыть конфигурационный файл **/etc/ssh/sshd_config** и задать значения для некоторых настроек:

```
sudo nano /etc/ssh/sshd_config
```

- Далее нам надо найти в файле четыре настройки - если они закомментированы, то раскомментируем их и установим им следующие значения:

```
# запрет аутентифицироваться суперпользователем
PermitRootLogin no

# разрешение аутентифицироваться с помощью публичного ключа
PubkeyAuthentication yes

# разрешение аутентифицироваться с логина и пароля
PasswordAuthentication yes

# запрет на использование пустых паролей
PermitEmptyPasswords no
```

Окончательный запуск SSH-сервера

- Мы установили новые значения для SSH, но пока они не были применены - для этого следует заставить SSH-сервер их перечитать. Для это воспользуемся следующей командой:

```
sudo systemctl reload ssh
```

- Несмотря на то, что наши изменения вступили в силу, мы пока не можем подключиться к серверу удаленно - у нас действует firewall, следовательно закрыты все порты. Чтобы исправить это недоразумение, нам следует открыть порт на котором работает SSH-сервер - по умолчанию это 22 порт:

```
sudo ufw allow 22
```

Теперь наш сервер полностью настроен и готов принимать удаленные соединения!

Соединение с удаленным сервером с помощью имени пользователя и пароля

- Чтобы соединиться с удаленным сервером, нам нужен какой-нибудь другой компьютер (можно даже виртуальный), на котором будут установлена клиентская программа, позволяющая запрашивать SSH соединения. На современный системах на ядрах Linux и Windows эта программа доступна изначально, но если по какой-то причине ее нет, то можно скачать, например, утилиту Putty, которая умеет работать с SSH: https://www.putty.org/.

- Когда мы определимся с клиентской программой, мы должны перейти в командную строку операционной системы (или командную строку Putty) и соединиться с удаленным сервером. Предположим, что IP сервера - **192.168.0.99**, а имя нашего пользователя внутри сервера - **user**. Чтобы осуществить соединение, выполним следующую команду:

```
ssh user@192.168.0.99
```

- На данном шаге нам будет предложено ввести пароль пользователя **user** - если мы введем его правильно, то окажемся внутри командной строки удаленного сервера и сможем производить обычные операции - устанавливать программы, редактировать файлы и т.д.

Соединение с удаленным сервером с помощью публичного ключа (подготовка)

- Ввод пароля при каждом подключении может оказаться утомительным занятием, поэтому существует альтернатива. Мы генерируем на локальном компьютере пару ключей (публичный и закрытый) и кладем их внутри директории нашего пользователя во внутренней директории **.ssh**. Затем идем на удаленный сервер и копируем публичный ключ во внутреннюю директорию **.ssh** нашей удаленной пользовательской директории - в файл **authorized_keys**.

- Для начала нам нужно сгенерировать ключи на нашем локальном компьютере. Это осуществляется командой **ssh-keygen** - после параметра **-t** надо указать алгоритм для создания ключей (будем использовать **ecdsa**), а после параметра **-b** размер ключа в байтах (укажем **521**):

```
ssh-keygen -t ecdsa -b 521
```

Далее подтверждаем создание ключей и размещение их в директории **.ssh** - на предложение ввести ключевое слово (своеобразный пароль для ключа - его надо будет вводить каждый раз) не будем вводить ничего, сейчас мы не хотим себя этим нагружать. В результате в директории **.ssh** должны находиться два файла - **id_ecdsa** и **id_ecdsa.pub**. Копируем содержимое **id_ecdsa.pub** и отправляемся на наш удаленный сервер.

Соединение с удаленным сервером с помощью публичного ключа (реализация)

- На удаленном сервере направляемся в директорию нашего пользователя (например, **/home/user**), создаем директорию **.ssh**, внутри нее создаем файл **authorized_keys** и любым способом записываем внутрь него содержимое **id_ecdsa.pub**. Теперь все готово для удаленного подключения без пароля:

```
ssh user@192.168.0.99
```

  При первом подключении нам потребуется подтвердить наши действия - для этого надо в командной строке просто написать слово **yes**. Во все последующие разы мы будет подключаться к серверу без всяких паролей и подтверждений.

- Если наш закрытый ключ (**id_ecdsa**) лежит не внутри нашей пользовательской директории в директории **.ssh**, а в неком другом месте, то при подключении мы обязаны явно указать путь к файлу ключа с помощью параметра **-i**:

```
ssh -i /path/to/id_ecdsa user@192.168.0.99
```

- Важно помнить, что соединение с помощью ключа возможно в том случае, если на удаленном сервере в настроечном файле **/etc/ssh/sshd_config** значение настройки **PubkeyAuthentication** установлено как **yes**.

Копирование файлов при помощи SSH

- Для копирования файла (например, example.txt) на удаленный сервер используется команда **scp**:

```
scp ./example.txt user@192.168.0.99:/home/user/example.txt
```

- Для обратной операции - копирования файла (например, server.txt) с удаленного сервера на локальный компьютер нам надо поменять местами аргументы:

```
scp user@192.168.0.99:/home/user/server.txt ./server.txt
```

- Также мы можем копировать на сервер директории (например, local) со всем содержимым - для этого надо подставить команде **scp** параметр **-r**:

```
scp -r ./local user@192.168.0.99:/home/user/local
```

- Как и в случае с файлом, при копировании директории (например, remote) с сервера на компьютер нам просто надо поменять аргументы местами:

```
scp -r user@192.168.0.99:/home/user/remote ./remote
```

Установка и настройка WEB-сервера (Nginx)

Введение

- Подключение по протоколу SSH - отличный способ для работы внутри сервера, однако это не может обеспечить реализацию главной цели - предоставление услуг нашим клиентам. Таким образом, нам нужна еще одна программа, которая сможет отображать содержимое для посетителей сети - т.н. web-сервер. Именно web-сервер будет принимать запросы из сети, а затем либо отдавать статическую информацию (картинки, документы и т.д.), либо передавать управление некой серверной технологии (например, языку программирования - NodeJs, PHP, Python и т.д.), а затем отдавать результат работы этой технологии.

- Существует несколько популярных web-серверов, которые используются при развертывании приложений. Классическим вариантом является использование **Apache**, который появился уже в 1995 году. Также есть специализированные сервера, например, **Tomcat** (появился в 1999 году), который заточен под язык программирования Java. Однако наиболее современным и развивающимся web-сервером является **Nginx** (появился в 2004 году), который показывает отличные результаты при работе с высокой нагрузкой, менее требователен к ресурсам по сравнению с другими серверами, а также умеет пробрасывать запросы другим серверам (при этом осуществляя балансировку нагрузки).

Установка и запуск WEB-сервера Nginx

- Для установки web-сервера Nginx надо воспользоваться стандартной утилитой операционной системы Ubuntu под названием **apt**:

```
sudo apt install -y nginx
```

- Чтобы запустить Nginx воспользуемся главным процессом Ubuntu - **systemd** (и его специальной утилитой **systemctl**), который возьмет на себя ответственность за непрерывную работу нашего web-сервера:

```
sudo systemctl start nginx
```

- Для проверки того, что наш web-сервер успешно запустился и активен, надо выполнить нижележащую команду - если в ее выводе будет слово **active** - Nginx функционирует нормально:

```
sudo systemctl status nginx
```

- Важно понимать, что Nginx пока не виден в сети, т.к. мы пока не открыли (с помощью firewall) порты для его работы - это будет сделано только после завершения полной настройки web-сервера.

Конфигурационная структура Nginx

- Важнейший файл с настройками web-сервера Nginx находится по следующему адресу - **/etc/nginx/nginx.conf**. Позже мы разберем его подробнее, но в данный момент упомянем, что в этом файле устанавливается пользователь, от имени которого запускаются процессы web-сервера, устанавливаются основные ограничения на количество соединений, а также подключаются дополнительные модули и конфигурации отдельных сайтов.

- Конфигурации отдельных сайтов находятся в двух директориях. Как правило, сначала файл (с любым удобным для нас названием) создается в директории **/etc/nginx/sites-available**. В файле прописываются директории, которые будут доступны для просмотра со стороны сети, возможные перенаправления запросов на обработчики уже упоминавшихся серверных технологий, а также такие вещи как путь к SSL сертификату, доменное имя сайта и прослушиваемые порты. Когда заполнение файла закончено, в директории **/etc/nginx/sites-enabled** на него создается символическая ссылка, которую подхватывает основной конфигурационный файл **/etc/nginx/nginx.conf**. Однако для этого надо перечитать настройки web-сервер с помощью нижеприведенной команды:

```
sudo systemctl reload nginx
```

Обзор главного конфигурационного файла **/etc/nginx/nginx.conf**

- Параметр **user** указывает на пользователя, от имени которого обслуживаются процессы, обрабатывающие запросы клиентов. По умолчанию **www-data**.
- **worker_processes** назначает количество процессов, которые обрабатывают запросы. По умолчанию **auto** (кол-во равняется кол-ву ядер процессов).
- **worker_connections** внутри блока **events** указывает на количество соединений, которые одномоментно может обслуживать один процесс - по умолчанию **768**.
- Далее идет блок **http**, где перечисляются настройки **http** соединений, например:
  - **include /etc/nginx/mime.types;** - подключение типов файлов
  - **access_log /var/log/nginx/access.log;** - путь к файлу лога соединений
  - **error_log /var/log/nginx/error.log;** - путь к файлу лога ошибок
  - **gzip on;** - включение (**on**) или выключение (**off**) сжатия ответа сервера
  - **include /etc/nginx/sites-enabled/*;** - путь к конфигурациям сайтов

Упрощенный пример главного конфигурационного файла

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
}

http {
    include /etc/nginx/mime.types;

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    gzip on;

    include /etc/nginx/sites-enabled/*;
}
```

Обзор конфигурационного файла конкретного сайта (основные директивы)

- Как уже было сказано, конфигурационные файлы сайтов хранятся внутри **/etc/nginx/sites-available**. Конфигурация любого сайта должна быть обернута в блок server { … }, внутри которого располагаются настройки сайта, например:
  - В настройке **listen** следует указать порт, на который должен приходит запрос к сайту. Можно указать порт как для IPv4, так и для IPv6 адреса.
  - В настройке **root** указывается путь к корневой директории сайта. Процессы сервера должны иметь соответствующие права на эту директорию.
  - В настройке **index** можно указать несколько названий файлов, которые будут показаны, если клиент обратится не к файлу, а к директории сайта.
  - В настройке **server_name** указываются имена сайта. Если указано _, то это означает имя по умолчанию (если другие не найдены, то применится это).
  - В настройке **try_files** можно перечислить файлы, которые будут показаны, если файл, запрошенный пользователем, не будет существовать.

Обзор конфигурационного файла конкретного сайта (блок **location**)

- Внутри блока server мы можем объявить блок location, в рамках которого можно указать поведение для специфического пути, например, вот так:
  - **location /static { … }** - перехватывает запросы, путь которых начинается с элемента **/static** - например, при помощи директивы **alias** задать серверный путь, который будет обслуживать такие запросы.
  - **location = /contacts { … }** - перехватывает запросы, путь которых совпадает со строкой **/contacts**. Внутри блока можно перенаправить пользователя по другому пути при помощи директивы **rewrite** (после нее указываем путь).
  - **location ~ \.php$ { … }** - перехват запросов регистрозависимым регулярным выражением - здесь перехватываются запросы, оканчивающиеся на строку “**.php**”. В блоке можно, например, задать заголовок (с помощью директивы **add_header**, после которой идет название заголовка, а потом значение).
  - **location ~* \.png$ { … }** - перехват запросов регистронезависимым регулярным выражением - в данном случае перехватываются запросы, оканчивающиеся на строку “**.png**”, “**.PNG**”, “**.pNg**” и т.д. Внутри блока можно запретить доступ всем при помощи директивы **deny all**.

Пример конфигурационного файла конкретного сайта

```
server {
    listen 80;
    listen [::]:80;
    root /var/www/html;
    index index.html index.php;
    server_name mysite.com www.mysite.com;

    location /static {
        alias /srv/staticfiles;
    }

    location = /contacts {
        rewrite /feedback;
    }

    location ~* \.png {
        deny all;
    }
}
```

Окончательный запуск WEB-сервера Nginx

- После того, как мы настроим конфигурационный файл нашего сайта (допустим, что он называется site), мы должны создать символическую ссылку на него в директории **/etc/nginx/sites-enabled**:

```
sudo ln -s /etc/nginx/sites-available/mysite /etc/nginx/sites-enabled/mysite
```

- Затем мы должны перечитать конфигурацию web-сервера:

```
sudo systemctl reload nginx
```

- В самом конце мы обязаны открыть 80 порт для входящих соединений:

```
sudo ufw allow 80
```

- Теперь сайты нашего web-сервера доступны по сети без всяких ограничений. Если у нас куплено доменное имя, например, mysite.com и правильно установлены DNS параметры, то мы можем набрать в браузере адрес http://mysite.com и увидеть то, что хранится на сервере.

Установка SSL сертификата

Введение

- В современном мире крайне остро стоит вопрос безопасности. Посетитель сайта хочет, чтобы его данные не были украдены или подделаны, была бы сохранена тайна переписки, а также присутствовала уверенность в том, что сайт, которым он пользуется, является настоящим, а не поддельным. Чтобы решить все эти проблемы, в сетях используются так называемые SSL сертификаты. Сайт, имеющий этот сертификат, предоставляет его браузеру клиента, затем браузер проверяет подлинность этого сертификата на неких авторитетных ресурсах (адреса ресурсов вшиты в сам браузер). Если проверка прошла успешно - между браузером и сервером устанавливается шифрованное соединение, по которому относительно безопасно передаются данные.

- В данном разделе мы посмотрим, как подключить к сайту (в контексте сервера Nginx) упомянутый SSL сертификат, а также как его правильно настроить. Мы не будем приобретать сертификат у некой большой организации, кроме того мы не будем использовать специальную программу **certbot**, чтобы подписать наш сайт с помощью сервиса **Let's Encrypt** (мы сделаем это в будущем). Вместо это мы создадим свой сертификат и сами подпишем его (с определенными оговорками браузер позволит его использовать) этого будет достаточно для знакомства с этой областью знаний.

Создание своего сертификата

- Для создания своего сертификата нам понадобится программа **openssl** - если у нас ее нет, то можно установить ее следующей командой:

```
sudo apt install -y openssl
```

- Теперь следует приступить непосредственно к созданию - нам понадобится нижележащая команда (все параметры этой команды будут разъяснены на следующей странице):

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048
-keyout /etc/ssl/private/certificate.key -out
/etc/ssl/certs/certificate.crt
```

- После ввода команды нам потребуется ввести дополнительную информацию - код нашей страны, название региона, город, имя организации, подразделение, наше имя и электронную почту. В результате у нас будут созданы два файла - **certificate.key** (закрытый ключ, с помощью которого создан сертификат и который используется для подтверждения подлинности сертификата) и **certificate.crt** (непосредственно сам сертификат).

Описание параметров команды **openssl** при создании своего сертификата

- **openssl req** - запрос на создание SSL сертификата
- **-x509** - параметр указывает, что сертификат будет подписан нами, а не какой-то другой организацией
- **-nodes** - отключить запрос пароля при запуске сервера, ссылающегося на сертификат
- **-days 365** - срок годности сертификата
- **-newkey rsa:2048** - алгоритм генерации закрытого ключа (в данном случае RSA) и размер данного ключа (в данном случае - 2048 бит)
- **-keyout /etc/ssl/private/certificate.key** - путь для файла, который будет содержать наш новый закрытый ключ
- **-out /etc/ssl/certs/certificate.crt** - путь для файла, который будет содержать наш новый SSL сертификат

Подключение сертификата к сайту

- Теперь мы должны подключить наши новые сертификаты в настройках сайта в контексте сервера Nginx. Для этого надо прослушивать 443 порт (для IPv4 и IPv6) - как мы помним, это осуществляется при помощи директивы **listen**. Затем мы должны указать путь к сертификату с помощью директивы **ssl_certificate**, а также путь к закрытому ключу с помощью директивы **ssl_certificate_key**. Также надо выключить сжатие, т.к. оно может вызвать проблемы с безопасностью:

```
server {
    listen 80;
    listen [::]:80;
    listen 443 ssl;
    listen [::]:443 ssl;

    gzip off;
    ssl_certificate     /etc/ssl/certs/certificate.crt;
    ssl_certificate_key /etc/ssl/private/certificate.key;

    root /var/www/html;
    index index.html index.php;
    server_name mysite.com www.mysite.com;

    location /static {
        alias /srv/staticfiles;
    }
}
```

Завершение внедрения SSL сертификата

- Для того, что изменения вступили в силу, нам в очередной раз надо перечитать настройки сервера:

```
sudo systemctl reload nginx
```

- Теперь мы можем зайти на наш сайт по адресу https://mysite.com (заметим, что теперь мы используем **https**, а не **http** - это значит, что мы хотим использовать TLS протокол и запрашиваем SSL сертификат). Т.к. мы установили сертификат, подписанный нами, в первый раз браузер скажет, что не может ничего распознать. Мы подтвердим, что все равно хотим посетить сайт и в дальнейшем у нас не будет проблем с навигацией.

Развертывание сайта на PHP

Введение

- Все наши предыдущие настройки сайта и сервера сводились к тому, чтобы просто отдавать клиенту статичные файлы, которые находятся в корневом каталоге сервера или в каталогах, внутренних по отношению к корневому каталогу. Однако в современном мире этого мало - нам может понадобится обработать запрос клиента специфическим образом, показать динамические значения (например, время на данный момент), а также сохранить данные в базу данных (что подразумевает, установление соединения с этой базой). Очевидно, что статические файлы этого делать не умеют.

- Для решения обозначенной проблемы нам необходимо использовать некую технологию, которой сервер перенаправит запрос, технология его обработает должным образом, потом передаст результат серверу, а сервер вернет полученные данные клиенту. Как правило, в качестве серверных технологий используются языки программирования. Одним из самых популярных языков в рамках сайтостроительства надо признать PHP, который настолько сросся с этой сферой, что у web-серверов (в том числе у Nginx) есть заранее подготовленные конфигурационные файлы для него.

Установка и развертывание PHP

- Обычно PHP отсутствует в стандартных репозиториях Ubuntu, поэтому сначала нам надо добавить PHP репозиторий вручную (этого не надо делать, если репозиторий уже был добавлен ранее):

```
sudo add-apt-repository ppa:ondrej/php
sudo apt update
```

- Теперь мы должны скачать последнюю версию PHP (на данный момент это 8.2), кроме того для работы с Nginx нам понадобится специальная версия FPM - (FastCGI Process Manager):

```
sudo apt install -y php8.2-fpm
```

- После установки **php8.2-fpm** мы должны запустить его в как фоновый процесс (который в свою очередь породит свои процессы, готовые к обработке запросов перенаправленных web-сервером). Можно понять, что **php8.2-fpm** тоже по-сути является сервером, только в данном случае сервером приложения:

```
sudo systemctl start php8.2-fpm
```

Режимы работы PHP-FPM

- PHP-FPM может работать в двух режимах:
    - Во-первых, PHP-FPM может слушать порт на неком компьютере, и именно на этот порт WEB-сервер сможет перенаправлять запросы из сети - это можно назвать TCP режимом. В данном случае web-сервер и PHP полностью отделены друг от друга, они могут даже находится на разных компьютерах или в разных контейнерах (если мы используем Docker). Данный подход дает определенные преимущества при масштабировании и балансировке нагрузки, однако слегка снижает быстродействие по причине того, что данные передаются по сети (пусть даже и по локальной сети).
    - Во-вторых, PHP-FPM для своей работы может использовать UNIX-сокет (по сути, файловый дескриптор, который разные процессы могут использовать для чтения и записи). В данном случае PHP-FPM и web-сервер должны находится на одном компьютере - иначе они не будут иметь доступа к UNIX-сокету. Данный подход имеет преимущество в скорости по сравнению с предыдущим вариантом, однако проигрывает в гибкости настройки и ограничивает в масштабировании (в этом случае мы не сможем установить PHP на 100 других компьютеров, чтобы снизить нагрузку).

Настройка PHP-FPM

- Файл настроек PHP-FPM находится по адресу **/etc/php/8.2/fpm/pool.d/www.conf.** Зайдем в этот файл с правами суперпользователя:

```
sudo nano /etc/php/8.2/fpm/pool.d/www.conf
```

- Далее найдем в открытом файле директиву **listen**. Если мы хотим использовать PHP-FPM в режиме TCP, то в качестве значения этой директивы надо установить номер порта (как правило устанавливают 9000):

```
listen = 9000
```

Если же мы хотим заставить PHP-FPM работать в режиме UNIX-сокета, то для директивы listen нужно подставить путь к файлу этого сокета:

```
listen = /run/php/php8.2-fpm.sock
```

- После выбора режима работы мы сохраняем файл настроек и заставляем PHP-FPM перечитать свои конфигурации:

```
sudo systemctl reload php8.2-fpm
```

Подключение PHP-FPM к WEB-серверу Nginx

- Теперь нам надо перейти в настроечный файл нашего сайта и прописать там правила для использования PHP:

```
sudo nano /etc/nginx/sites-available/mysite
```

Далее внутри блока **server** мы должны создать новый блок **location**, который должен обрабатывать пути, которые заканчиваются на “.php” (расширение PHP файлов). Уже внутри блока **location** нам следует подгрузить специальный файл настроек PHP, которое предлагает сам Nginx (**include snippets/fastcgi-php.conf;**). Затем если мы запустили PHP-FPM в TCP режиме на 9000 порту, то прописываем следующую директиву: **fastcgi_pass 127.0.0.1:9000;**. Если же наш PHP-FPM работает в режиме UNIX-сокета, то директива выглядит вот так: **fastcgi_pass unix:/run/php/php8.2-fpm.sock;**. После этого сохраняем файл и перечитываем конфигурацию сервера:

```
sudo systemctl reload nginx
```

Пример конфигурационного файла сайта с подключенным PHP (в TCP режиме)

```
server {
    listen 80;
    listen [::]:80;
    listen 443 ssl;
    listen [::]:443 ssl;

    gzip off;
    ssl_certificate     /etc/ssl/certs/certificate.crt;
    ssl_certificate_key /etc/ssl/private/certificate.key;

    root /var/www/html;
    index index.php index.html;
    server_name mysite.com www.mysite.com;

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass 127.0.0.1:9000;
    }
}
```

Создание PHP скриптов

- С этого момента мы можем создавать **php** скрипты в директории нашего сайта. Например, создадим файл **time.php**, показывающий время на данный момент:

```
<?php

date_default_timezone_set('Europe/Riga');
echo date('Y-m-d H:i:s');
```

- При переходе по ссылке https://mysite.com/time.php мы не увидим этот код, а увидим результат выполнения этого кода - т.е. рижское время.

Развертывание сайта на Nodejs

Введение

- PHP - замечательная вещь для разработки сайтов, но и она не идеальна. В последнее время всё большую популярность в серверной разработке обретает язык программирования JavaScript (в контексте технологии Nodejs). Главными преимуществами JavaScript является, во-первых, отсутствие блокировок при операциях ввода/вывода, что позволяет одновременно обслуживать большее количество соединений, и, во-вторых, более приспособленная модель взаимодействия с постоянными соединениями, что полезно для игр, приложений реального времени и прочих программ, использующих мгновенные обновления.

- Как уже упоминалось, для того, чтобы работать с JavaScript на сервере, нам потребуется технология Nodejs. Кроме того, если мы захотим установить дополнительные библиотеки, нам будет нужен пакетный менеджер NPM:

```
sudo apt install -y nodejs npm
```

Простейшее приложение на Nodejs

- Создадим простейшее приложение на Nodejs - для этого нам потребуется новая директория, например, **/var/www/js**. Перейдем в эту директорию и создадим файл **index.js**. Затем в качестве зависимости загрузим фреймворк express.js, который используется для разработки web-приложений:

```
npm install express
```

- Затем сохраним в файле **index.js** следующий код, который будет возвращать разное содержимое для корневого пути сайта и пути **/blog**:

```
const express = require('express')
const app = express()
const port = 8000

app.get('/', (req, res) => res.send('This is the main page.'))

app.get('/blog', (req, res) => res.send('Here will be articles.'))

app.listen(port)
```

Запуск приложения Nodejs (простая версия)

- Если мы запустим наше приложение обычным для Nodejs способом, то оно будет слушать 8000 порт и успешно отвечать на запросы со стороны сети:

```
node index.js
```

  Однако как только мы закроем командную строку, работа нашего приложения автоматически прекратится. Она прекратится даже в том случае, если мы запустили бы эту программу в фоновом режиме, т.е. в конце был бы добавлен амперсанд **&**.

- Чтобы предотвратить автоматическое закрытие приложения, нам необходима фоновая программа (демон), которая будет отвечать за запуск приложения, а также за его перезапуск, если произойдет ошибка. В рамках Nodejs такой программой является **pm2** - ее мы рассмотрим на следующей странице.

Запуск приложения Nodejs (продвинутая версия с помощью pm2)

- Для установки демона pm2 нам необходимо выполнить следующую команду (она установит программу-демон глобально, а не только для нашего проекта):

```
sudo npm install pm2@latest -g
```

- Теперь мы можем запустить наше приложение правильно - для начала перейдем в директорию с приложением, а затем выполним эту команду:

```
pm2 start index.js
```

- После того, что мы сделали, можно уже не бояться, что наше приложение будет автоматически остановлено после закрытия командной строки или в каком-либо другом случае. Важно отметить, что **pm2** предоставляет множество других удобных опций - например, мониторинг запущенных процессов (**pm2 ls**), автоматический запуск приложений при старте операционной системы (**sudo pm2 startup**) и т.д. Подробнее с этими опциями можно познакомится на сайте разработчиков: https://pm2.keymetrics.io/docs/usage/quick-start/

Запуск pm2 в качестве службы

- Сам по себе демон pm2 умеет перезапускать приложение в случае ошибки, однако он бессилен, если будет перезагружена операционная система. Однако мы можем зарегистрировать демон в качестве службы, которая, как уже ранее говорилось, автоматически стартует при инициализации операционной системы. Сначала создадим файл службы при помощи следующей команды:

```
pm2 startup systemd
```

Далее нам будет сказано запустить определенную команду от имени суперпользователя - копируем эту команду и запускаем.

- Теперь мы должны зафиксировать список работающих Nodejs приложений:

```
pm2 save
```

- Остановим все работающие с помощью pm2 приложения, а потом запустим их вновь, но уже с помощью службы (если мы создавали службу от имени пользователя user, то служба будет называться pm2-user):

```
pm2 kill
sudo systemctl start pm2-user
```

Подключение приложения Nodejs к WEB-серверу Nginx

- Теперь нам надо перейти в настроечный файл нашего сайта и прописать там правила для использования NodeJs приложения:

```
sudo nano /etc/nginx/sites-available/mysite
```

В рамках традиционного блока **server** нам надо создать блок **location**, который будет слушать запросы, начинающиеся с корневой директории (**/**). Внутри следует расположить перебрасывание запроса на 8000 порт, который слушает Node.js (при помощи директивы **proxy_pass**), ряд заголовков, которые понадобятся приложению для работы (**Host** - доменное имя приложения, **Upgrade** и **Connection** - используются для работы на уровне web-сокетов, **X-Real-IP** - IP адрес клиента, **X-Forwarded-For** - IP адреса тех прокси-серверов, через которые прошел запрос), а также версию протокола HTTP (**proxy_http_version**). Также не будет лишним применить директиву **proxy_redirect off;** - она запрещает nginx самовольно перенаправлять клиентский запрос, если проксируемое приложение вернуло специальный заголовок **Location** - в это случае перенаправлением должна заниматься сама клиентская программа.

Пример конфигурационного файла сайта с подключенным Nodejs приложением

```
server {
    listen 80;
    listen [::]:80;
    listen 443 ssl;
    listen [::]:443 ssl;

    gzip off;
    ssl_certificate     /etc/ssl/certs/certificate.crt;
    ssl_certificate_key /etc/ssl/private/certificate.key;

    server_name mysite.com www.mysite.com;

    location / {
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $http_host;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        proxy_redirect off;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

Окончательный запуск Nodejs приложения

- Теперь нам надо перечитать конфигурации Nginx, чтобы наши изменения вступили в силу:

```
sudo systemctl reload nginx
```

- С этого момента мы можем посещать наш Nodejs сайт по тем адресам, которые установлены в конфигурации - https://mysite.com и http://mysite.com.

Развертывание сайта на Python

Введение

- В качестве еще одной популярной технологии разработки сайтов можно назвать язык программирования Python. Python является языком общего назначения, поэтому он не столь удобен для web-разработки как JavaScript и PHP, которые создавались в первую очередь для сайтостроительства. Однако популярность Python в мире (т.е. будет легко найти ответ на любой вопрос) и множество дополнительных библиотек и удобных фреймворков (например, Django) все же позволяют рассматривать этот язык программирования в качество хорошего варианта для создания web-приложений.

- Если мы используем Ubuntu, то нам не надо устанавливать Python - он у нас уже есть (**python3**). Однако сначала мы должны создать директорию для сайта и включить виртуальное окружение (чтобы не засорять наш компьютер специфическими версиями библиотек, которые потом станут конфликтовать):

```
sudo mkdir /srv/python-site
sudo chown www-data:www-data /srv/python-site
cd /srv/python-site
python3 -m venv .venv
source ./.venv/bin/activate
```

Установка фреймворка Django

- Самым популярным фреймворком для разработки приложений на Python является Django. Он не входит в стандартную библиотеку, поэтому нам следует загрузить этот фреймворк из сети Интернет с помощью установщика **pip**:

```
pip install django
```

- Теперь необходимо в нашей директории на базе Django создать новый проект, который будет называться **app**. Для этого следует выполнить следующий код:

```
django-admin startproject app .
```

- После этого установим дополнительный проект под названием, например, **blog** (будет отвечать за отображение страниц для простых пользователей):

```
python3 manage.py startapp blog
```

- В самом конце нам следует запустить миграции для базы данных (по умолчанию используется файловая база данных **sqlite**, которую не надо устанавливать):

```
python3 manage.py migrate
```

Подготовка конфигурационного файла

- Далее в корневой директории нашего проекта создадим конфигурационный файл **.env**, где будем хранить системные настройки - пока ограничимся только **SECRET_KEY** для сессий (ключом должна быть строка с разными символами):

```
SECRET_KEY=6-!mchkzzk5)0uh23r(o90%t7qqqb!mr(o-43+0lfkjpyv3
```

- Затем мы должны установить модуль Python под названием python-dotenv, который позволит использовать значения конфигурационного файла .env внутри нашего Django приложения:

```
pip install python-dotenv
```

Настройка обработчиков запросов и путей запросов в контексте Django

- В директории **blog** в файле **views.py** мы создадим два обработчика запросов:

```
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return HttpResponse("This is main page.")

def pages(request):
    return HttpResponse("Here will be articles.")
```

- Затем в этой же директории создадим файл **urls.py** и подключим обработчики в шаблоны для путей:

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.index, name="index"),
    path("pages/", views.pages, name="pages"),
]
```

Подключение путей проекта blog к основному приложению Django

- Теперь перейдем в основную директорию приложения **app** и сначала настроим подключение путей. Для этого откроем файл **urls.py** и приведем его в следующий вид (т.е. подключим пути приложения **blog**):

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('blog.urls'))
]
```

Настройка главного системного файла Django (подгрузка зависимостей)

- Пришла пора настроить самые главные конфигурации, которые находятся в директории **app** в файле **settings.py**. В самом начале нам надо подгрузить две дополнительных вещи - модуль **os** (для взаимодействия с переменными окружения), а также функцию **load_dotenv** из модуля **dotenv** (для загрузки переменных окружения). Затем сразу же вызовем функцию **load_dotenv**, чтобы как раз подгрузить переменные окружения.

- Теперь начало файла **settings.py** должно выглядеть следующим образом:

```
from pathlib import Path
from dotenv import load_dotenv
import os

# take environment variables from .env.
load_dotenv()
```

Настройка главного системного файла Django (установка значений)

- Далее найдем в файле **settings.py** переменную **SECRET_KEY** (используется для шифрования сессии) и установим ей значение из переменной окружения:

```
SECRET_KEY = os.environ['SECRET_KEY']
```

- Отыщем переменную **DEBUG** и установим ее значение в **False** (таким образом наше приложение будет переведено из режима разработки в боевой режим):

```
DEBUG = False
```

- В переменную **ALLOWED_HOSTS** впишем название тех хостов, по которых можно обращаться к нашему сайту - мы разрешим **localhost**, **127.0.0.1** и **mysite.com**:

```
ALLOWED_HOSTS = ['localhost', '127.0.0.1', 'mysite.com']
```

- Затем найдем переменную **INSTALLED_APPS** и добавим в нее настройки нашего внутреннего приложения **blog**:

```
INSTALLED_APPS = [
    …,
    'blog.apps.BlogConfig'
]
```

Завершение настройки главного системного файла и копирование статических файлов

- Последнее, что мы сделаем в файле settings.py - добавление совершенно новой переменной **STATIC_ROOT** - там будет хранится путь к директории, где будут находится статические файлы (**css**, **js** и т.д.):

```
STATIC_ROOT = BASE_DIR / 'static'
```

- Теперь сохраним файл **settings.py** со всеми изменениями, а затем создадим в корне проекта ту самую директорию **static**, где будут хранится все статические файлы:

```
mkdir static
```

- Затем наконец скопируем все доступные нам в данный момент статические файлы (пока это только файлы административной панели, но также это могут быть файлы отдельных внутренних приложений) следующей командой:

```
python3 manage.py collectstatic
```

Создание администратора для Django проекта

- Наше приложение Django по умолчанию располагает удобной административной панелью, с помощью которой можно управлять сайтом. Однако перед использованием этой панели нам следует создать администратора. Это осуществляется при помощи следующей команды:

```
python manage.py createsuperuser
```

После этого нам будет предложено ввести имя администратора, его почту, а затем пароль. Если все пройдет успешно, то администратор будет создан. Когда наше приложение заработает, административная панель будет доступна по адресу http://mysite.com/admin.

Развертывание сервера приложений Gunicorn (введение)

- По сути, наше приложение уже готово к запуску. Мы может инициализировать его по некому порту и пробросить этот порт в сеть. Однако это будет не очень безопасно, т.к. встроенный сервер приложений Django предназначен только для разработки, а не для использования в реальных условиях. Поэтому мы должны сначала установить специальный сервер приложений Gunicorn, с помощью этого сервера запустить приложение Django, а в самом конце настроить web-сервер Nginx таким образом, чтобы он перенаправлял запросы из сети на Gunicorn.

- Gunicorn можно установить с помощью pip - менеджера зависимостей Python:

```
pip install gunicorn
```

Создание конфигурационного файла для службы **gunicorn**

- Самым правильным решением будет запустить gunicorn в качестве службы, которая будет автоматически загружаться при старте операционной системы и за работой которой будет следить главный процесс **systemd**. Для этого в директории **/etc/systemd/system** следует создать файл **gunicorn.service**:

```
sudo nano /etc/systemd/system/gunicorn.service
```

- Внутри созданного файла должно быть три секции:
  - Во-первых, **[Unit]** - содержит два пункта, **Description** (описание) и **After** (зависимости), который в качестве значения получит **network.target**, т.е. для работы этой службы нужна полная загрузка всех сетевых служб.
  - Во-вторых, **[Service]** - содержит настройки служб - **User** (пользователь службы), **Group** (группа службы), **WorkingDirectory** (изначальная рабочая директория службы), **ExecStart** (команда запуска процесса службы).
  - В-третьих, секция **[Install]** - нам потребуется настройка **WantedBy**, которая будет равняться **multi-user.target**. Это означает, что служба должна запуститься в тот момент, когда в операционной системе появится хотя бы одна сессия пользователя (еще до загрузки графического интерфейса).

Пример конфигурационного файла для службы **gunicorn**

Файл **/etc/systemd/system/gunicorn.service**

```
[Unit]
Description=gunicorn daemon
After=network.target

[Service]
User=www-data
Group=www-data
WorkingDirectory=/srv/python-site
ExecStart=/srv/python-site/.venv/bin/gunicorn \
          --access-logfile - \
          --workers 3 \
          --bind 127.0.0.1:8000 \
          app.wsgi:application

[Install]
WantedBy=multi-user.target
```

Запуск службы **gunicorn**

- Пришла пора запустить нашу новую службу. Однако перед этим нам надо перечитать настройки **systemd**, чтобы он полноценно “осознал” новую службу:

```
sudo systemctl daemon-reload
```

- Теперь запускаем службу **gunicorn**, которая будет слушать 8000 порт (так прописано в конфигурационном файле):

```
sudo systemctl start gunicorn
```

Подключение **gunicorn** к WEB-серверу Nginx

- Все, что нам осталось - перейти в настроечный Nginx файл нашего сайта и прописать там правила для использования gunicorn приложения:

```
sudo nano /etc/nginx/sites-available/mysite
```

Как и в случае с Nodejs, в рамках блока **server** нам надо создать блок **location**, который будет слушать запросы, начинающиеся с корневой директории (**/**). Внутри следует расположить перебрасывание запроса на 8000 порт, который будет слушать **gunicorn** (при помощи директивы **proxy_pass**), ряд заголовков, которые понадобятся приложению для работы (**Host** - доменное имя приложения, **X-Real-IP** - IP адрес клиента, **X-Forwarded-For** - IP адреса тех прокси-серверов, через которые прошел запрос и **X-Forwarded-Proto** - версия протокола - **http** или **https** - иногда бывает полезно для внутреннего перенаправления). Также опять применим директиву **proxy_redirect off;** - она запрещает nginx самовольно перенаправлять клиентский запрос, если проксируемое приложение вернуло специальный заголовок **Location** - в это случае перенаправлением должна заниматься сама клиентская программа. Также надо создать блок **location**, который начинается со слова **/static/** - он будет напрямую отдавать статические файлы при помощи директивы **alias**.

Пример конфигурационного файла сайта с подключенным **gunicorn** приложением

```
server {
    listen 80;
    listen [::]:80;
    listen 443 ssl;
    listen [::]:443 ssl;

    gzip off;
    ssl_certificate     /etc/ssl/certs/certificate.crt;
    ssl_certificate_key /etc/ssl/private/certificate.key;

    server_name mysite.com www.mysite.com;

    location /static/ {
      alias /srv/python-site/static/;
    }

    location / {
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_set_header Host $http_host;
      proxy_set_header X-Real-IP $remote_addr;

      proxy_redirect off;
      proxy_pass http://127.0.0.1:8000;
    }
}
```

Окончательный запуск **gunicorn** приложения

- Опять перечитаем конфигурации Nginx, чтобы изменения вступили в силу:

```
sudo systemctl reload nginx
```

- С этого момента мы можем посещать наш Django сайт по тем адресам, которые установлены в конфигурации - https://mysite.com и http://mysite.com.