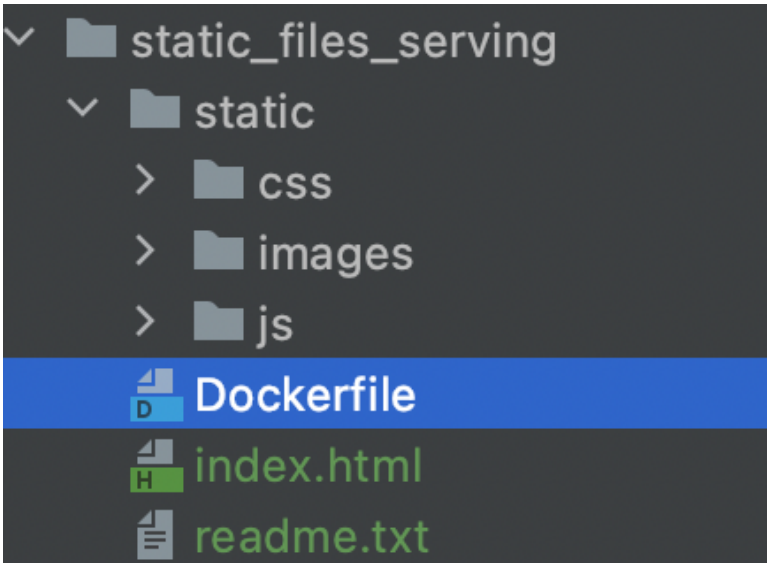
Урок 2. NGINX

Давайте разберёмся с тем, как работает NGNIX, и научимся отдавать статический контент.
Прежде всего условимся, что разворачивать сервер мы с вами будем в docker контейнерах — так проще и быстрее.

Начнём с того, что подготовим сайт — это уже известный нам сайт с отсчётом времени до Нового года. Поместим файлы в директории так, чтобы получилась следующая структура:



В самом докер-файле нам нужно выбрать базовый образ и скопировать файлы по нужным путям. Давайте это сделаем:

```
FROM nginx

COPY ./static /usr/share/nginx/html/static
COPY ./index.html /usr/share/nginx/html
```

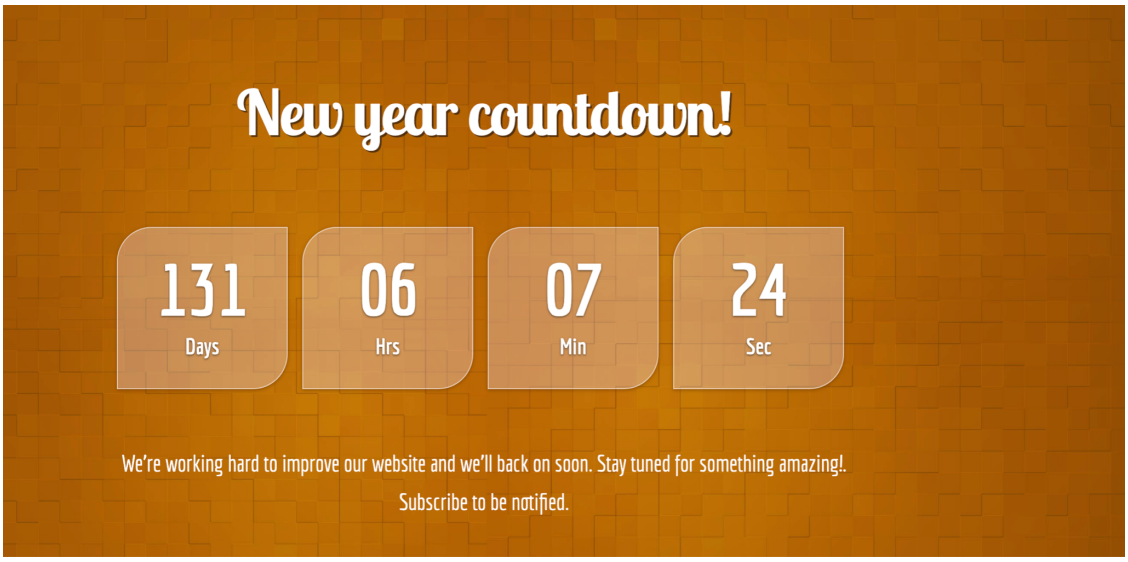
Команды нам уже знакомы: указываем, какой образ использовать как базовый, и копируем файлы по нужным путям внутри образа.

Мы выбрали этот образ, потому что он оптимально подходит текущей задаче — хостинг статики. Внутри уже установлен NGINX, выполнена минимальная конфигурация. Достаточно просто запустить контейнер, и всё будет работать. Далее по модулю мы с вами повторим все эти шаги самостоятельно, а пока давайте запустим наш сервер:

```
docker build . -f Dockerfile -t static_serving
docker run -ti -p 80:80 static_serving
```

Первой командой мы собираем образ и называем его, а второй запускаем образ. Обратим внимание на то, какой порт мы прокинули с докера на хотс — 80. Это стандартный порт для http-запросов. Браузер автоматически направляет весь http-трафик на этот порт, а веб-сервер по умолчанию ожидает на этом порту. Если бы у нас был HTPPS, то мы использовали бы порт 443.

Давайте попробуем открыть сайт в браузере:
http://localhost/index.html



Минимум усилий, а каков результат! В прошлый раз нам для этого потребовалось целое Flask-приложение. Не будем даже говорить о том, насколько мощнее и устойчивее текущее решение. В общем, на этом урок можно было бы и завершить, всё и так работает. Но, кажется, мы прошли мимо настройки сервера; давайте разберёмся с тем, почему всё работает. Для этого залезем внутрь имаджа. Откроем ещё один терминал и выполним:

```
docker run -ti static_serving bash
```

Эта команда позволит запустить внутри контейнера не дефолтную команду (entrypoint), а то что мы захотим, в данном случае bash. В результате мы сможем внимательно изучить внутренности контейнера.

Нас интересует директория /etc/nginx — перейдём в неё:

```
cd /etc/nginx/
```

Посмотрим, что в ней:

```
root@df781477e1f9:/etc/nginx#ls -la
total 36
drwxr-xr-x 3 root root 4096 Aug 17 09:16 .
drwxr-xr-x 1 root root 4096 Aug 22 14:59 ..
drwxr-xr-x 2 root root 4096 Aug 17 09:16 conf.d
-rw-r--r-- 1 root root 1007 Jul  6 14:59 fastcgi_params
-rw-r--r-- 1 root root 5290 Jul  6 14:59 mime.types
lrwxrwxrwx 1 root root   22 Jul  6 15:11 modules -> /usr/lib/nginx/modules
-rw-r--r-- 1 root root  648 Jul  6 15:11 nginx.conf
-rw-r--r-- 1 root root  636 Jul  6 14:59 scgi_params
-rw-r--r-- 1 root root  664 Jul  6 14:59 uwsgi_params
```

Видим тут ряд конфигов, давайте посмотрим на самый привлекательный — nginx.conf.

Давайте посмотрим, что там:

```
cat nginx.conf
```

```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;
```

Видим тут не такое уж и большое количество настроек. Давайте скопируем себе на хост этот файл. Для этого как раз есть удобная команда докера “cp”. Для неё потребуется имя запущенного контейнера. Чтобы узнать его, выполните docker ps:

```
docker cp crazy_cannon:/etc/nginx/nginx.conf ./
```

В результате мы сможем вытянуть себе на хост файл из имаджа и работать с ним в

IDE. Давайте откроем его:

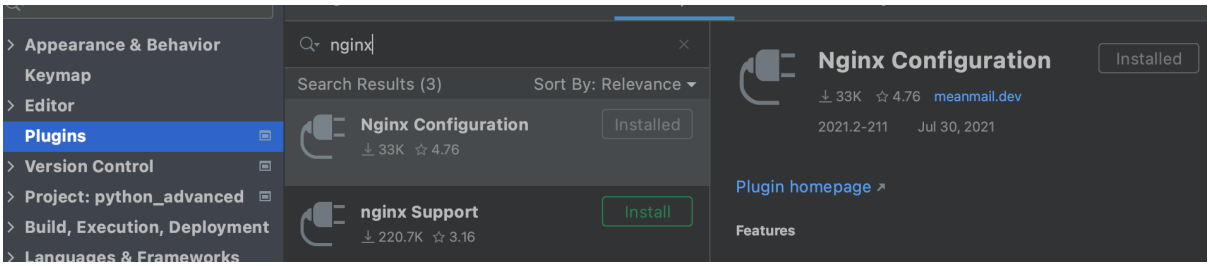
```
user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


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

Кстати, по умолчанию в PyCharm нет подсветки синтаксиса .conf-файлов, так что нам придётся установить плагин; для этого зайдём в «Настройки» — «Плагины» и там выберем Nginx Configuration.



Давайте изучим содержимое дефолтного конфига.

Первое, что мы видим:

```
user  nginx;
```

Так мы запускаем веб-сервер от менее привилегированного пользователя. Сделано это по соображениям безопасности.

Далее:

```
worker_processes  auto;
```

Так мы говорим серверу, сколько ядер он может использовать. Значение auto отдаёт все доступные, чтобы обеспечить лучшую производительность.

Далее две похожие строчки:

```
error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;
```

Первая говорит, куда писать логи. Параметр notice тут — это уровень логгирования. У NGINX они немного отличаются в нейминге. Этот уровень находится между дефолтным error (не очень подробным) и info (куда более подробным).
Вторая строка — тут мы говорим, куда записать пид запущенного процесса.
Это нужно, чтобы без кучи страшных grep вытянуть pid процесса сервера и завершить его, если есть такая необходимость.

```
events {
   worker_connections  1024;
}
```

Этой директивой мы устанавливаем максимальное количество одновременных соединений, которые может установить процесс-воркер.

И далее идут настройки непосредственно http-сервера. Но прежде чем погрузиться в них, давайте взглянем на самый конец конфига:

```
include /etc/nginx/conf.d/*.conf;
```

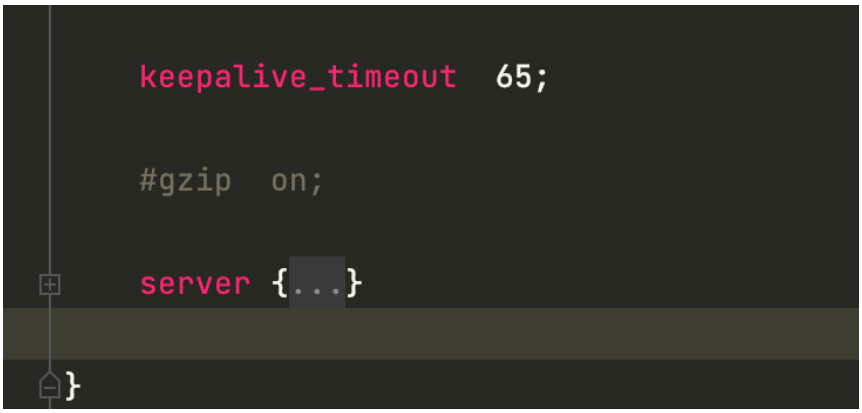
Нетрудно догадаться, что этой директивой мы подключаем ещё какие-то конфиги. Давайте копируем их себе на хост.

```
root@df781477e1f9:/etc/nginx# ls conf.d/
default.conf
```

```
docker cp crazy_cannon:/etc/nginx/conf.d/default.conf ./
```

Давайте не глядя скопируем содержимое файла и выставим его вместо этой директивы.

Получится в итоге что-то подобное:



И, кажется, тут понятно: на одном NGINX-сервере мы можем запускать сразу много разных приложений. Несколько таких директив server будут описывать разные сайты. А те, что находятся на уровне HTTP, будут общими для них всех.

Теперь, поскольку настроек много, а мы одни, давайте останавливаться только на важных, а ненужные будем удалять.

Первые 4 настройки оставим без изменений. Они конфигурируют логгирование и список поддерживаемых mimetypes.

```
sendfile        on;
```

Эта директива включает оптимизацию для более быстрой отдачи статики — оставляем.
Всё, что закомментировано, удаляем.

```
keepalive_timeout  65;
```

Сколько времени держать открытым соединение? В нашем случае некритично, но если бы отдавали динамическую страницу, то этот параметр помог бы закрыть соединение, если приложение зависло.

Далее:

```
 #gzip  on;
```

Тоже удалим. Эта настройка включает сжатие для отдаваемого контента. Вспомните о ней, когда будете оптимизировать время ответа от вашего приложения, а пока просто удалим её. Для любопытствующих будет ссылка под видео.
https://docs.nginx.com/nginx/admin-guide/web-server/compression/

```
server {
   listen       80;
   server_name  localhost;
```

Далее идёт настройка конкретного сервера. Директива listen указывает, на каком порту должен слушать сервер. И далее имя сервера; server_name смело можно удалять. Эта директива потребовалась бы нам, если бы на одном физическом сервере хостили несколько виртуальных: например, example.com и пример.рф. В этом случае в http-запросе обязательно будет содержаться header Host. По нему сервер поймёт, к какому «виртуальному серверу» идёт запрос.

Долой всё лишнее!
То, что закомментировано, тоже удаляем.

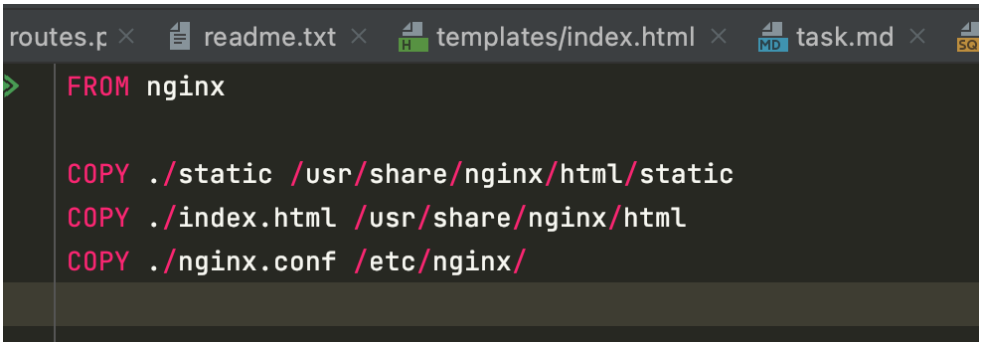
Наконец, самая важная часть:

```
location / {
    root   /usr/share/nginx/html;
    index  index.html index.htm;
}
```

Тут мы описываем пути нашего сервера и то, какой файл нужно вернуть по этому пути. Обратим внимание, что в прошлый раз, когда мы обращались к серверу, мы указали конкретный файл. И если мы просто обратимся на localhost, то попадём туда же — всё благодаря этой настройке. Буквально она говорит следующее: все, кто пришёл на рут сайта (/), должны получить в ответ из директории, что мы описали в root файл index.html

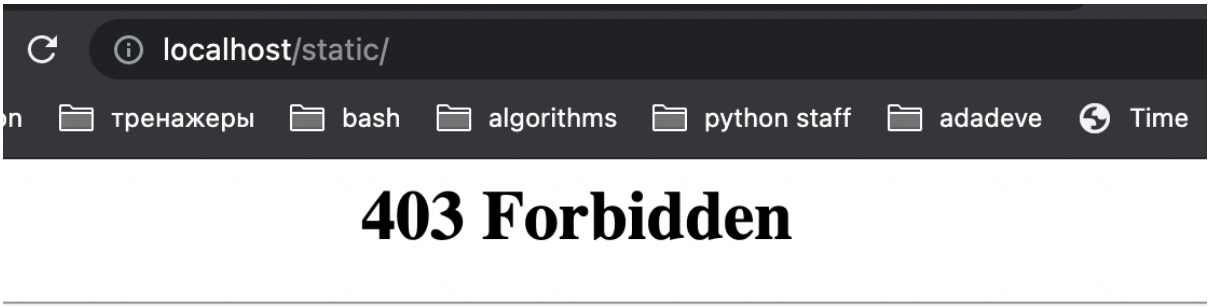
Дальнейшие настройки мы пока просто удалим, но просмотрите их самостоятельно: там о том, что вернуть пользователю, если случилась какая-то ошибка, и о том, как включить хостинг динамических страниц с языком php и перенаправить запрос на apache-сервер.

Давайте сохраним этот файл и добавим его в наш образ:



Не забываем пересоздать и перезапустить образ.
Идём на сайт проверить, как всё работает — без изменений.

Кстати, было бы удобно прямо в браузере получить список файлов из директории static.
Если мы обратимся к этой директории напрямую, то получим ошибку 403.

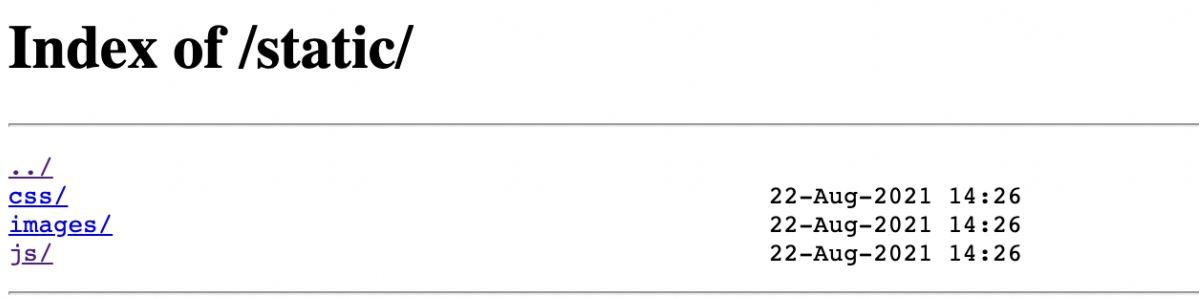


И в логах *2 directory index of "/usr/share/nginx/html/static/" is forbidden.

Давайте это дело разрешим.

```
server {
   listen       80;
   root   /usr/share/nginx/html;
   location / {
      autoindex on;
   }
}
```

Слегка перепишем директиву сервера и вытащим рут на уровень выше, а внутри «/» опишем директиву autoindex on — эта команда отрисует нам в браузере весь контент директории:



Вот так мы собрали себе веб-сервер для хостинга чего угодно на коленке. В следующих уроках мы разберёмся с тем, как подружить NGINX с Python и создавать динамические страницы.