Настройка HTTPS с помощью Nginx, Let’s Encrypt и Docker

Цели

- Запустить **nginx** в одном контейнере
- Запустить **другие проекты** в других контейнерах
- Научить **nginx** перенаправлять запросы с разных доменов на разные проекты
- Получить **ssl** сертификаты для всех проектов
- Убрать **www** из названия сайтов с помощью **nginx**

Перед началом

> Перед тем как начать, должно быть следующее

- Ubuntu сервер с открытыми портами `80` и `443`.
- Установленный `docker` и `docker-compose` на локальном компьютере и сервере.
- Зарегистрированные доменные имена. В этом руководстве я буду использовать `proj1.com` и `proj2.com` для двух проектов.
  - Запись `A` для `proj1.com` и `proj2.com`, указывает на публичный IP адрес нашего сервера.
  - Запись `A` для `www.proj1.com` и `www.proj2.com`, указывает на публичный IP адрес нашего сервера.

В этом руководстве будет использоваться

- nodejs
- create-react-app
- yarn
- docker
- docker-compose

В любой момент может понадобится

- `docker ps` - список запущенных контейнеров
- `docker network ls` - список всех активных и неактивных сетей созданных docker'ом
- `docker system prune` - очистка всех **остановленных контейнеров** и **неиспользуемых сетей**
- `docker-compose down` - выполняется из корня проекта. Останавливает запущенный контейнер.

Настройка на локальном компьютере

Этап 1. Создаем 2 `react` проекта

> В результате будет 2 простых проекта

В любом удобном месте на локальном компьютере создаем `react` проект `proj1` и `proj2`

`yarn create react-app proj1` `yarn create react-app proj2`

По очереди запускаем каждый из проектов `yarn start` и вносим маленькие изменения, чтобы видеть отличия проектов.

> Например в `src -> App.js` каждого проекта вместо `Edit <code>src/App.js</code> and save to reload.` (create-react-app версия 2.1.3) напишем `proj1` и `proj2` соответственно.

Этап 2. Упаковываем каждый проект в контейнер

> В результате каждый проект будет в отдельном контейнере. Все действия далее описываю только для одного проекта. Их нужно сделать для двух проектов.

Пишем инструкции для запуска контейнера

В корне проекта создаем `Dockerfile`

> **Dockerfile** - это инструкция по созданию контейнера

```
FROM mhart/alpine-node:11
  # Качаем node контейнер с docker hub
RUN yarn global add serve
  # Устанавливаем serve для сервирования нашего проекта
WORKDIR /app
  # Указываем главную папку в контейнере, в которой будет наш проект
COPY package.json yarn.lock ./
  # Копируем файлы package.json и yarn.lock в папку, указанную выше (не забываем ./)
RUN yarn
  # Устанавливаем зависимости
```

```
COPY . .
  # Копируем остальные файлы проекта в главную папку
RUN yarn build
  # Компилируем проект
CMD serve -s /app/build
  # Серверуем проект из новой папки build (по умолчанию порт 5000)
```

Далее в корне проекта создаем `docker-compose.yml`

> **docker-compose** - это инструкция по запуску контейнер(а/ов)

```
version: '3.7'
# Смотреть актуальную версию docker-compose синтаксиса на оф. сайте

services:
# Объявляем список сервисов
  proj1:
  # Указываем название сервиса
    container_name: proj1
    # Указываем название контейнера (не обязательно)
    build: .
    # Запускаем Dockerfile из корня проекта
    environment:
      - NODE_ENV=production
      # ENV переменная production сообщит yarn, что не нужно качать зависимости для
разработки, если такие имеются.
    ports:
    # Слева порт, который будет доступен на локальной машине. Справа - тот, к которому нужен
доступ в контейнере.
      - 8080:5000
      # 5000 - порт который по умолчанию открывает serve в Dockerfile
```

В корне проекта создаем `.dockerignore`

> **.dockerignore** - это список исключаемых из копирования `COPY . .` файлов и папок в `Dockerfile`

```
.git
node_modules
build
```

Запускаем и проверяем контейнер

Из корня проекта выполняем `docker-compose up`

> При первом запуске будут выполняться все инструкции `Dockerfile`, что займет какое-то время. Дальнейшие запуски, при отсутствии изменений в проекте, будут проходить быстрее.

Когда видим строку вида `proj1 | INFO: Accepting connections at http://localhost:5000`, можем проверить в браузере `localhost:8080`. Результатом должен быть `react` проект с надписью `proj1`.

В командной строке нажимаем `ctrl+c`, чтобы выйти из контейнера и проверяем проект `proj2`

Этап 3. Создаем 2 фейковых url для теста

> В результате будет 2 адреса, каждый из которых будет вести на `localhost`. Нужно для дальнейшей настройки `nginx`. **Этот шаг можно пропустить и продолжать действия на сервере**

В `hosts` файле на компьютере добавляем запись

```
127.0.0.1       proj1.com
127.0.0.1       proj2.com
```

Этап 4. Создаем `nginx` контейнер

В результате будет `nginx` контейнер, который будет работать на `80` порту

Создаем `docker-compose` для `nginx`

В удобном месте создаем папку `nginx`.

Затем в ней создаем файл `docker-compose.yml`

```
version: '3.7'

services:
  nginx:
    container_name: nginx
    image: nginx:latest
    # Вместо Dockerfile берем готовый nginx (:latest - последняя версия) с docker hub
    volumes:
      - ./data/nginx.conf:/etc/nginx/conf.d/default.conf
    ports:
      - 80:80
      # Порт 80 будет доступен как в контейнере, так и снаружи.
```

> **volumes** - позволяет использовать файлы из локального компьютера не копируя их в контейнер. В данном случае мы берем файл `nginx.conf` из папки `data`, который мы создадим дальше, в папке `nginx` на локальном компьютере и "говорим" контейнеру, что это файл который находится здесь `/etc/nginx/conf.d/default.conf`

Создаем `nginx.conf` для `nginx`

В папке `nginx` создаем папку `data` в которой создаем файл `nginx.conf` (название и путь должны соответствовать указанным в `docker-compose`)

```
server {
  return  301 http://google.com;
}
```

> Самый простой `conf`, для проверки работы контейнера.

Запускаем `nginx` контейнер

В папке `nginx` выполняем `docker-compose up`. В браузере при переходе на `localhost` мы должны попадать на сайт google.

> Останавливаем контейнер и переходим далее

Этап 5. Связываем контейнеры между собой

> В результате `nginx` контейнер сможет перенаправлять запросы в другие контейнеры. Действия описанные далее для `proj1` нужно выполнять для каждого проекта.

Меняем `docker-compose` файл в `nginx` проекте

В папке `nginx` в файле `docker-compose` под блоком `services` пишем новый блок `networks`

```
networks:
# Блок для объявления внутренних docker сетей, которые запустятся с этим файлом
  nginx_net:
  # Название сети для этого контейнера
    name: nginx_net
    # Название сети для внешних контейнеров
```

В блоке `services` в подблок `nginx` добавляем

```
networks:
  nginx_net:
  # Подключаемся к новой сети в этом контейнере
```

Результат:

```
version: '3.7'

services:
  nginx:
    container_name: nginx
    image: nginx:latest
    networks:
      nginx_net:
    volumes:
      - ./data/nginx.conf:/etc/nginx/conf.d/default.conf
    ports:
      - 80:80

networks:
  nginx_net:
    name: nginx_net
```

Меняем `docker-compose` файл в `react` проекте

В папке `proj1` в файле `docker-compose` удаляем

```
ports:
  - 8080:5000
```

> Контейнер по прежнему слушает на порту `5000`, но уже не открывает его для локального компьютера. Этот порт теперь доступен только контейнерам в той-же сети.

Под блоком `services` пишем

```
networks:
  nginx_net:
    external: true
    # Сообщает о том, что сеть nginx_net находится не в этом контейнере.
```

В блоке `services` в подблок `proj1` добавляем

```
networks:
  nginx_net:
```

Результат:

```
version: '3.7'

services:
  proj1:
    container_name: proj1
    build: .
    networks:
      nginx_net:
    environment:
      - NODE_ENV=production

networks:
  nginx_net:
    external: true
```

Меняем `nginx.conf` файл

Удаляем все, что писали для примера и пишем

```
server {
  server_name proj1.com;

  location / {
    resolver 127.0.0.11;
    set $project http://proj1:5000;

    proxy_pass $project;
```

```
  }
}

server {
  server_name proj2.com;

  location / {
    resolver 127.0.0.11;
    set $project http://proj2:5000;

    proxy_pass $project;
  }
}
```

> Каждый блок `server` отвечает за отдельные задачи.
>
> `server_name` определяет с какого именно домена пришел запрос.
>
> `location /` указывает на то, что нас устраивает как `proj1.com`, так и что угодно после `/`.
>
> `proxy_pass` перенаправляет запрос на наш контейнер. **proj1** в `http://proj1:5000` доступен благодаря названию контейнера в `docker-compose -> services`.
>
> `resolver 127.0.0.11;` и `set $project http://proj1:5000;` нужны для того, чтобы контейнер мог запустится даже при нерабочем контейнере `proj1`. Можно было написать и `proxy_pass http://proj2:5000;` без переменных, но в этом случае, если не запущен контейнер `proj1`, `nginx` не запустится и выдаст ошибку.

Запускаем все контейнеры `proj1`, `proj2` и `nginx`. В результате, вводя в браузере `proj1.com` и `proj2.com` мы попадаем на наши 2 проекта.

Этап 6. Переносим контейнеры на сервер

Если, все описанное выше работает, можно переносить наши проекты на сервер заменив все названия проектов, доменов в `nginx.conf` `proj1` и `proj2` на необходимые названия и домены. Для удобства я продолжу использовать `proj1` и `proj2`.

В `nginx.conf` в `server_name` добавляем `www` записи. В итоге строка будет выглядеть так

```
server_name proj1.com www.proj1.com;
```

Убедившись, что на сервере все работает, можно приступать к следующей части.

Настройка на сервере

> Далее, много информации берется от сюда

Этап 1. Подключаем `certbot` контейнер для запуска с `nginx`

> В результате в папке `nginx`, `docker-compose up` будет запускать 2 контейнера. `nginx` и `certbot`

В `services -> nginx -> ports` добавляем

```
- 443:443
# Открываем порты для ssl соединения
```

В `services` добавляем

```
certbot:
# Новый контейнер, который запуститься вместе с nginx
  container_name: certbot
  image: certbot/certbot
  # Образ берется с docker hub
  networks:
```

```
    nginx_net:
    # Подключаем к той-же сети, что и остальные контейнеры
```

> Если мы запустим контейнер сейчас, будет ошибка, так как `certbot` не может провести первичную настроку в docker контейнере.

Этап 2. Добавляем пути к сертификатам

> В конце мы запустим скрипт, который сгенерирует, сначала, фейковые сертификаты, для запуска `nginx`, а затем и настоящие. В этой части мы прописываем пути по которым `nginx` и `certbot` смогут их увидеть.

В `services -> nginx` добавляем

```
- ./data/certbot/conf:/etc/letsencrypt
- ./data/certbot/www:/var/www/certbot
```

В `services -> certbot` добавляем

```
volumes:
- ./data/certbot/conf:/etc/letsencrypt
- ./data/certbot/www:/var/www/certbot
```

Результат

```
version: '3.7'

services:
  nginx:
    container_name: nginx
    image: nginx:latest
    networks:
```

```
      nginx_net:
    volumes:
      - ./data/nginx.conf:/etc/nginx/conf.d/default.conf
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    ports:
      - 80:80
      - 443:443
  certbot:
    container_name: certbot
    image: certbot/certbot
    networks:
      nginx_net:
    volumes:
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot

networks:
  nginx_net:
    name: nginx_net
```

Этап 3. Редактируем `nginx`

> Здесь мы создаем финальную конфигурацию `nginx`. В примере будет 1 `server` блок отвечающий за `proj1`. Соответственно, сколько проектов, столько и `server` блоков.

Существует множество конфигураций `nginx`. Здесь приведена одна из них.

```
server {
  listen 80;
  listen 443 ssl;
  # Слушаем на портах 80 и 443
  server_name proj1.com www.proj1.com;
  # Этот сервер блок выполняется при этих доменных именах
```

```
  ssl_certificate /etc/letsencrypt/live/proj1.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/proj1.com/privkey.pem;
  # ssl_certificate и ssl_certificate_key - необходимые сертификаты
  include /etc/letsencrypt/options-ssl-nginx.conf;
  ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
  # include и ssl_dhparam - дополнительные, рекомендуемые Let's Encrypt, параметры

  # Определяем, нужен ли редирект с www на без www'шную версию
  if ($server_port = 80) { set $https_redirect 1; }
  if ($host ~ '^www\.') { set $https_redirect 1; }
  if ($https_redirect = 1) { return 301 https://proj1.com$request_uri; }

  location /.well-known/acme-challenge/ { root /var/www/certbot; }
  # Путь по которому certbot сможет проверить сервер на подлинность

  location / {
    resolver 127.0.0.11;
    set $project http://proj1:5000;

    proxy_pass $project;
  }
}
```

Этап 4. Завершаем настройку `docker-compose.yml` для `nginx` и `certbot`

В `services -> nginx` добавляем

```
restart: unless-stopped
# Перезапустит контейнер в непредвиденных ситуациях
command: '/bin/sh -c ''while :; do sleep 6h & wait $${!}; nginx -s reload; done & nginx -g
"daemon off;"'''
# Перезапустит nginx контейнер каждые 6 часов и подгрузит новые сертификаты, если есть
```

В `services -> certbot` добавляем

```
restart: unless-stopped
entrypoint:  "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h & wait $${!}; done;'"
# Проверяет каждые 12 часов, нужны ли новые сертификаты
```

Результат

```
version: '3.7'

services:
  nginx:
    container_name: nginx
    image: nginx:latest
    restart: unless-stopped
    networks:
      nginx_net:
    volumes:
      - ./data/nginx.conf:/etc/nginx/conf.d/default.conf
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    ports:
      - 80:80
      - 443:443
    command: '/bin/sh -c ''while :; do sleep 6h & wait $${!}; nginx -s reload; done & nginx -g "daemon off;"'''

  certbot:
    container_name: certbot
    image: certbot/certbot
    restart: unless-stopped #+++
    networks:
      nginx_net:
    volumes:
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h & wait $${!}; done;'"
```

```
networks:
  nginx_net:
    name: nginx_net
```

Этап 5. Получаем сертификаты

Находясь в папке `nginx` вводим

```
curl -L https://raw.githubusercontent.com/dancheskus/nginx-docker-ssl/master/init-letsencrypt.sh > init-letsencrypt.sh
```

Затем открываем полученный `init-letsencrypt.sh` и вписываем:

- вместо `example.com` свои домены через пробел
- email
- меняем пути, если меняли их в папке
- `staging` временно ставим 1 (для теста)

`chmod +x init-letsencrypt.sh`

`sudo ./init-letsencrypt.sh`

Выполнив последнюю команду в папке `nginx -> data` появятся новые папки с сертификатами необходимые для работы `nginx` и `certbot` Если в тестовом режиме все получилось, то ставим `staging=0` и повторяем процедуру.

Готово

> `docker-compose run --rm --entrypoint "certbot certificates" certbot` - позволяет проверить зарегистрированные сертификаты и узнать их срок годности.

Revision #1
Created 4 April 2023 03:32:14 by 3err0
Updated 4 April 2023 03:32:28 by 3err0