

Docker

- Containers
 - samba
 - rsync-server
 - postgres
 - hass
 - mysql
 - nginx
 - photoprism
 - resilio-sync
 - vaultwarden
 - gitea
 - traefik
 - tt-rss
 - mongo
 - bookstack
 - webdav
- Настройка HTTPS с помощью Nginx, Let's Encrypt и Docker
- Установка Docker
- Очистка
- Disable IPv6 in docker compose

Containers

samba

```
version: '3.8'

services:
  samba:
    image: dperson/samba
    restart: always
    networks:
      samba_net:
        ipv4_address: 10.1.0.20
    volumes:
      - /opt/samba:/mnt/share:rw
    command: >
      -s "public;/mnt/share;yes;no;yes;all"
      -p
    tmpfs:
      - /tmp
    environment:
      TZ: "Asia/Irkutsk"
      USERID: "1000"
      GROUPID: "1000"
      ANON: "yes"
      ANON_UID: "65534"
      ANON_GID: "65534"
    ports:
      - "137:137/udp"
      - "138:138/udp"
      - "139:139/tcp"
      - "445:445/tcp"
    deploy:
      resources:
        limits:
          cpus: '0.5'
          memory: '512M'
```

```
placement:
  constraints: [node.role == worker]
```

```
networks:
```

```
  samba_net:
```

```
    driver: macvlan
```

```
    driver_opts:
```

```
      parent: enp3s0
```

```
  ipam:
```

```
    driver: default
```

```
    config:
```

```
      - subnet: 10.1.0.0/21
```

```
        gateway: 10.1.0.1
```

rsync-server

```
version: "2.1"
services:
  rsync-server:
    image: apnar/rsync-server
    container_name: rsync-server
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Asia/Irkutsk
      - USERNAME=user
      - PASSWORD=user
      - VOLUME=/backup
      - ALLOW=0.0.0.0/0
    volumes:
      - backup: /backup
    ports:
      - 873:873
    restart: unless-stopped

volumes:
  backup:
    external: true

networks:
  default:
    name: sync
```

postgres

```
version: '3.7'
services:
  pgsql:
    container_name: postgres
    image: timescale/timescaledb:latest-pg15
    restart: always
    environment:
      - POSTGRES_PASSWORD=password
      - POSTGRES_USER=root
      - POSTGRES_DB=postgres
    volumes:
      - /opt/pgsql/data : /var/lib/postgresql/data
    ports:
      - "5432: 5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U root -d postgres"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 10s
    restart: unless-stopped
    deploy:
      resources:
        limits:
          cpus: '4'
          memory: 16G

networks:
  default:
    name: databases
```

hass

```
version: '3'
services:
  homeassistant:
    container_name: homeassistant
    image: "ghcr.io/home-assistant/home-assistant: stable"
    volumes:
      - /opt/hass: /config
      - /etc/localtime: /etc/localtime:ro
    restart: unless-stopped
    privileged: true
    network_mode: host
```

mysql

```
version: '3.9'

services:

  db:
    container_name: mysql
    image: mariadb
    restart: always
    command: mysqld --port=3306 --innodb-strict-mode=1 --innodb-buffer-pool-size=256M --
transaction-isolation=READ-COMMITTED --character-set-server=utf8mb4 --collation-
server=utf8mb4_unicode_ci --max-connections=512 --innodb-rollback-on-timeout=OFF --innodb-
lock-wait-timeout=120
    volumes:
      - /opt/mysql/data:/var/lib/mysql
      - /opt/mysql/etc:/etc/mysql/conf.d
    ports:
      - 3306:3306
    environment:
      MARIADB_AUTO_UPGRADE: "1"
      MARIADB_INITDB_SKIP_TZINFO: "1"
      MARIADB_ROOT_PASSWORD: "password"

  adminer:
    container_name: adminer
    image: adminer
    restart: always
    ports:
      - 8086:8080

networks:
  default:
    name: databases
    driver: bridge
```


external: true

nginx

```
version: '3'

services:
  webserver:
    image: nginx:latest
    container_name: nginx
    ports:
      - 80:80
      - 443:443
    command: '/bin/sh -c ''while :; do sleep 6h & wait $$(!); nginx -s reload; done & nginx -g
"daemon off;''''
    restart: unless-stopped
    volumes:
      - /opt/nginx/conf/: /etc/nginx/conf.d/:ro
      - /opt/nginx/certbot/www: /var/www/certbot/
      - /opt/nginx/certbot/conf: /etc/nginx/ssl/
      - /opt/nginx/certbot/conf: /etc/letsencrypt/
  certbot:
    image: certbot/certbot:latest
    container_name: certbot
    volumes:
      - /opt/nginx/certbot/www: /var/www/certbot/:rw
      - /opt/nginx/certbot/conf: /etc/letsencrypt/:rw
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h & wait
$$(!); done;'"

networks:
  default:
    name: webserver
```

photoprism

```
version: '3.5'

services:
  photoprism:
    container_name: photo
    image: photoprism/photoprism:latest
    security_opt:
      - seccomp:unconfined
      - apparmor:unconfined
    ports:
      - "2344:2342"
    environment:
      PHOTOPRISM_INIT: "http"
      PHOTOPRISM_UID: ${UID:-1000}
      PHOTOPRISM_GID: ${GID:-1000}                # group id
      PHOTOPRISM_ADMIN_USER: "admin"              # admin username
      PHOTOPRISM_ADMIN_PASSWORD: "photoprism"     # initial admin password (minimum 8
characters)
      PHOTOPRISM_AUTH_MODE: "password"            # authentication mode (public,
password)
      PHOTOPRISM_SITE_URL: "https://photo/"      # server URL in the format
"http(s)://domain.name(:port)/(path)"
      PHOTOPRISM_SITE_CAPTION: "Latest"
      PHOTOPRISM_SITE_DESCRIPTION: "Tags and finds pictures without getting in your way!"
      PHOTOPRISM_SITE_AUTHOR: "@3err0"
      PHOTOPRISM_DEBUG: "false"
      PHOTOPRISM_READONLY: "false"
      PHOTOPRISM_EXPERIMENTAL: "false"
      PHOTOPRISM_HTTP_MODE: "release"
      PHOTOPRISM_HTTP_HOST: "0.0.0.0"
      PHOTOPRISM_HTTP_PORT: 2342
      PHOTOPRISM_HTTP_COMPRESSION: "gzip"        # improves transfer speed and
bandwidth utilization (none or gzip)
```

```

PHOTOPRISM_DATABASE_DRIVER: "mysql"
PHOTOPRISM_DATABASE_SERVER: "10.0.0.1:3306"
PHOTOPRISM_DATABASE_NAME: "photoprism"
PHOTOPRISM_DATABASE_USER: "root"
PHOTOPRISM_DATABASE_PASSWORD: "password"
PHOTOPRISM_DISABLE_CHOWN: "false"      # disables updating storage permissions via
chmod and chown on startup
PHOTOPRISM_DISABLE_BACKUPS: "true"     # disables backing up albums and photo metadata
to YAML files
PHOTOPRISM_DISABLE_WEBDAV: "true"      # disables built-in WebDAV server
PHOTOPRISM_DISABLE_SETTINGS: "false"   # disables settings UI and API
PHOTOPRISM_DISABLE_PLACES: "false"     # disables reverse geocoding and maps
PHOTOPRISM_DISABLE_EXIFTOOL: "false"   # disables creating JSON metadata sidecar files
with ExifTool
PHOTOPRISM_DISABLE_TENSORFLOW: "false" # disables all features depending on TensorFlow
PHOTOPRISM_DETECT_NSFW: "false"        # automatically flags photos as private that MAY
be offensive (requires TensorFlow)
PHOTOPRISM_UPLOAD_NSFW: "false"        # allows uploads that MAY be offensive (no
effect without TensorFlow)
PHOTOPRISM_RAW_PRESETS: "false"        # enables applying user presets when converting
RAW files (reduces performance)
PHOTOPRISM_THUMB_FILTER: "lanczos"     # resample filter, best to worst: blackman,
lanczos, cubic, linear
PHOTOPRISM_THUMB_UNCACHED: "true"      # enables on-demand thumbnail rendering (high
memory and cpu usage)
PHOTOPRISM_THUMB_SIZE: 2048            # pre-rendered thumbnail size limit (default
2048, min 720, max 7680)
# PHOTOPRISM_THUMB_SIZE: 4096          # Retina 4K, DCI 4K (requires more storage);
7680 for 8K Ultra HD
PHOTOPRISM_THUMB_SIZE_UNCACHED: 7680   # on-demand rendering size limit (default 7680,
min 720, max 7680)
PHOTOPRISM_JPEG_SIZE: 7680             # size limit for converted image files in pixels
(720-30000)
PHOTOPRISM_JPEG_QUALITY: 85            # a higher value increases the quality and file
size of JPEG images and thumbnails (25-100)
TF_CPP_MIN_LOG_LEVEL: 0               # show TensorFlow log messages for development
working_dir: "/photoprism"
volumes:
- photo: /photoprism/originals
# - ". /storage: /photoprism/storage"

```

networks:

 default:

 name: media

volumes:

 photo:

 external: true

resilio-sync

```
version: "2.1"
services:
  resilio-sync:
    image: lscr.io/linuxserver/resilio-sync:latest
    container_name: resilio-sync
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Asia/Irkutsk
    volumes:
      - backup: /sync/backup
      - photo: /sync/photo
      - /opt/sync/config: /config
      - /opt/sync/data: /sync
    ports:
      - 8888: 8888
      - 55555: 55555
    restart: unless-stopped

volumes:
  backup:
    external: true
  photo:
    external: true

networks:
  default:
    name: sync
```

vaultwarden

```
version: "3"

services:
  vaultwarden:
    image: vaultwarden/server:latest
    container_name: vaultwarden
    restart: unless-stopped
    ports:
      - 9445:80 #map any custom port to use (replace 8445 not 80)
    volumes:
      - /opt/bitwarden:/data:rw
    environment:
      - ADMIN_TOKEN=token
      - WEBSOCKET_ENABLED=false
      - SIGNUPS_ALLOWED=true
      # - SMTP_HOST=smtp.yandex.ru
      # - SMTP_FROM=mail@yandex.ru
      # - SMTP_PORT=465
      # - SMTP_SECURITY=starttls
      # - SMTP_USERNAME=mail@yandex.ru
      # - SMTP_PASSWORD=password
      - DOMAIN=https://password

networks:
  default:
    name: apps
```

gitea

```
version: "3"

networks:
  apps:
    external: false

services:
  server:
    image: gitea/gitea:1.18.3
    container_name: gitea
    environment:
      - USER_UID=1000
      - USER_GID=1000
      - GITEA__database__DB_TYPE=mysql
      - GITEA__database__HOST=10.0.0.1:3306
      - GITEA__database__NAME=git
      - GITEA__database__USER=root
      - GITEA__database__PASSWD=password
    restart: always
    networks:
      - apps
    volumes:
      - /opt/gitea:/data
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "3000:3000"
```

- Gitea with traefik

```
version: "3"

services:
  gitea:
```


image: gitea/gitea:1.18.3

container_name: gitea

environment:

- USER_UID=1000
- USER_GID=1000
- GITEA__database__DB_TYPE=mysql
- GITEA__database__HOST=10.10.10.2:3306
- GITEA__database__NAME=base
- GITEA__database__USER=user
- GITEA__database__PASSWD=pass

restart: always

networks:

- web

volumes:

- /opt/gitea:/data
- /etc/timezone:/etc/timezone:ro
- /etc/localtime:/etc/localtime:ro

labels:

- "traefik.enable=true"
- "traefik.http.routers.gitea.rule=Host(`gitea_url`)"
- "traefik.http.services.gitea-websecure.loadbalancer.server.port=3000"
- "traefik.http.routers.gitea.tls=true"
- "traefik.http.routers.gitea.tls.certresolver=http"

networks:

web:

external: true

traefik

```
version: '3'

services:
  traefik:
    image: traefik:latest
    container_name: traefik
    restart: unless-stopped
    security_opt:
      - no-new-privileges:true
    ports:
      - 80:80
      - 443:443
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /opt/traefik/data/traefik.yml:/traefik.yml:ro
      - /opt/traefik/ssl/acme.json:/acme.json
      - /opt/traefik/custom:/custom:ro
    networks:
      - web
      - internal
    labels:
      - 'traefik.enable=true'
      - 'traefik.http.routers.traefik.entrypoints=http'
      - 'traefik.http.routers.traefik.rule=Host(`traefik_url`)'
      - 'traefik.http.middlewares.traefik-auth.basicauth.users=root:password'
      - 'traefik.http.middlewares.traefik-https-redirect.redirectscheme.scheme=https'
      - 'traefik.http.routers.traefik.middlewares=traefik-https-redirect'
      - 'traefik.http.routers.traefik-secure.entrypoints=https'
      - 'traefik.http.routers.traefik-secure.rule=Host(`traefik_url`)'
      - 'traefik.http.routers.traefik-secure.middlewares=traefik-auth'
      - 'traefik.http.routers.traefik-secure.tls=true'
      - 'traefik.http.routers.traefik-secure.tls.certresolver=http'
```

```
- 'traefik.http.routers.traefik-secure.service=api@internal'

networks:
  web:
    external: true
  internal:
    external: false
```

- traefik.yml

```
api:
  dashboard: true

entryPoints:
  http:
    address: ":80"
  https:
    address: ":443"

providers:
  docker:
    endpoint: "unix:///var/run/docker.sock"
    exposedByDefault: false
  file:
    directory: /custom
    watch: true

certificatesResolvers:
  http:
    acme:
      email: mail@3err0.ru
      storage: acme.json
      httpChallenge:
        entryPoint: http
```

- custom_service.yaml in custom directory

```
http:
  routers:
    custom-secure:
      rule: "Host(`url_service`)"
```

```
    service: "custom-service"
    entrypoints: ["https"]
    middlewares:
      - "custom-https-redirect"
    tls:
      certResolver: "http"
  psw:
    rule: "Host(`url_service`)"
    entrypoints: ["http"]
    middlewares:
      - "custom-https-redirect"
    service: "custom-service"

middlewares:
  custom-https-redirect:
    redirectScheme:
      scheme: "https"
      permanent: true

services:
  custom-service:
    loadBalancer:
      servers:
        - url: "http://service:80"
```

tt-rss

TTRss with traefik and postgresql database

```
version: '3'
services:
  ttrss:
    image: wangqiru/ttrss
    container_name: ttrss
    environment:
      DB_NAME: rss
      DB_USER: user
      DB_PASS: pass
      DB_HOST: 10.10.10.2
      DB_PORT: 5432
      SELF_URL_PATH: https://rss_url
    volumes:
      - /opt/ttrss/plugins:/var/www/plugins.local
    networks:
      - web
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.ttrss.rule=Host(`rss_url`)"
      - "traefik.http.services.ttrss.loadbalancer.server.port=80"
      - "traefik.http.routers.ttrss.tls=true"
      - "traefik.http.routers.ttrss.tls.certresolver=http"
    restart: always
networks:
  web:
    external: true
```

mongo

```
version: '3.9'

services:
  mongo:
    container_name: mongo
    image: mongo:latest
    restart: always
    volumes:
      - /opt/mongoddb/data: /data/db
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: user
      MONGO_INITDB_ROOT_PASSWORD: pass

networks:
  default:
    name: databases
```

bookstack

- Bookstack with traefik

```
version: '3'
services:
  bookstack:
    image: ghcr.io/linuxserver/bookstack
    container_name: bookstack
    restart: always
    environment:
      - DB_HOST=10.10.10.2:3306
      - DB_DATABASE=base
      - DB_USERNAME=user
      - DB_PASSWORD=password
      - APP_URL=https://docs.3err0.ru
      - STORAGE_TYPE=local
    networks:
      - web
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.bookstack.rule=Host(`docs.3err0.ru`)"
      - "traefik.http.services.bookstack.loadbalancer.server.port=80"
      - "traefik.http.routers.bookstack.tls=true"
      - "traefik.http.routers.bookstack.tls.certresolver=http"
networks:
  web:
    external: true
```

webdav

```
version: '3'
services:
  webdav:
    image: bytemark/webdav
    container_name: webdav
    restart: unless-stopped
    environment:
      AUTH_TYPE: Basic
      USERNAME: admin
      PASSWORD: password
      SERVER_NAMES: webdav.site
    networks:
      - web
    security_opt:
      - no-new-privileges:true
    volumes:
      - /opt/webdav:/var/lib/dav
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.webdav.entrypoints=http"
      - "traefik.http.routers.webdav.rule=Host(`webdav.site`)"
      - "traefik.http.middlewares.webdav-https-redirect.redirectscheme.scheme=https"
      - "traefik.http.routers.webdav.middlewares=webdav-https-redirect"
      - "traefik.http.routers.webdav-secure.entrypoints=https"
      - "traefik.http.routers.webdav-secure.rule=Host(`webdav.site`)"
      - "traefik.http.routers.webdav-secure.tls=true"
      - "traefik.http.routers.webdav-secure.tls.certresolver=http"
      - "traefik.http.routers.webdav-secure.service=webdav"
      - "traefik.http.services.webdav.loadbalancer.server.port=80"
      - "traefik.docker.network=web"
    networks:
      web:
        external: true
```


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 попасть 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 127.0.0.1;
    set $project http://proj1:5000;

    proxy_pass $project;
  }
}

server {
  server_name proj2.com;
```

```
location / {  
    resolver 127.0.0.1;  
    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 127.0.0.1;` и `set $project http://proj1:5000;` нужны для того, чтобы контейнер мог запускаться 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 приступить 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 настройку 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.

Установка Docker

This documentation shows how to install docker from the docker ppa

Create users and groups

First create a user and group `cloud`

```
useradd -u 1002 -m cloud
```

Add the user `cloud` to the sudo group

```
usermod -a -G sudo cloud
```

Add a `docker` group

```
groupadd docker
```

Add the user `cloud` to the docker group

```
usermod -aG docker cloud
```

Gibe the user the bash shell

```
usermod --shell /bin/bash cloud
```

basic dependencies

Install some basic dependencies

```
apt update && apt upgrade -y && apt install vim screen lsof apt-transport-https ca-  
certificates curl software-properties-common -y
```

Ubuntu 20.04

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-  
common
```

docker repository configuration

Add the repository key of the docker repository

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Now add the apt configuration for the repository

```
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

You could also alternatively add the `testing` repository instead of the `stable` repository

```
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) test"
```

install docker

```
apt update && apt install docker-ce docker-ce-cli containerd.io -y
```

install docker-compose

Get the latest docker-compose

```
curl -L https://github.com/docker/compose/releases/download/v2.12.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

Set the proper rights

```
chmod +x /usr/local/bin/docker-compose
```

And check if `docker-compose` works

```
docker-compose --version
```

This should give a response like this

```
docker-compose version 1.25.0-rc2, build 661ac20e
```

Очистка

Some times you want to clean up your docker environment after experimenting, this shows you how you can do it.

Prune images

```
docker image prune -a
```

Prune containers

```
docker container prune
```

Prune volumes

```
docker volume prune
```

Prune networks

```
docker network prune
```

Prune everything

```
docker system prune --volumes
```

Stop the container(s) using the following command:

```
docker-compose down
```

Delete all containers using the following command:

```
docker rm -f $(docker ps -a -q)
```

Delete all volumes using the following command:

```
docker volume rm $(docker volume ls -q)
```

Restart the containers using the following command:

```
docker-compose up -d
```

Disable IPv6 in docker compose

I have a docker-compose project that I am trying to run on my server, which does not have IPv6 enabled. Whenever I try to run the container, I get the following error message:

```
nginx: [emerg] socket() [::]:80 failed (97: Address family not supported by protocol)
```

I figured that is because IPv6 is not enabled on my server (it is managed by a third party, so I can't touch that), so I tried disabling IPv6 for docker-compose, so far without any luck.

I tried adding

```
sysctls:  
  net.ipv6.conf.all.disable_ipv6: 1
```

on my config file, but then I received the following error Error response from daemon: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: error during container init: open /proc/sys/net/ipv6/conf/all/disable_ipv6: no such file or directory: unknown

How can I disable IPv6 in docker-compose, either for this particular container or system-wide to not have issues like this?

This is my current config

```
container_name: cont-nginx  
networks:  
  - cont  
image: nginx:latest  
depends_on:  
  - cont-app  
restart: always  
ports:  
  - "880:880"  
  - "4443:4443"  
sysctls:
```

```
- net.ipv6.conf.all.disable_ipv6=1
volumes:
- . /nginx.conf: /etc/nginx/nginx.conf
```

networks: cont: driver: bridge

Solution : Disabling IPv6 for the docker's network should do the job:

networks: cont: driver: bridge enable_ipv6: false Also, maybe you should consider removing this from your nginx conf

listen [::]:80; because [::] is for IPv6.