**Docker**

Оглавление

[Место расположения 2](#_Toc118287259)

[Информация в целом 2](#_Toc118287260)

[IMAGE 3](#_Toc118287261)

[DIVE 3](#_Toc118287262)

[DOCKERFILE 4](#_Toc118287263)

[build 4](#_Toc118287264)

[Multistaging build 5](#_Toc118287265)

[NETWORK 5](#_Toc118287266)

[Команды 5](#_Toc118287267)

[MOUNT FILES AND DISKS 6](#_Toc118287268)

[VOLUMES 6](#_Toc118287269)

[volume in Dockerfile 6](#_Toc118287270)

[BIND папки 6](#_Toc118287271)

[TRPFS 7](#_Toc118287272)

[Копирование файлов и папок 7](#_Toc118287273)

[В контейнер 7](#_Toc118287274)

[Из контейнера 7](#_Toc118287275)

[DOCKER COMPOSE 7](#_Toc118287276)

[Install 8](#_Toc118287277)

[Команды 8](#_Toc118287278)

[Примеры: 10](#_Toc118287279)

# CONTAINER

docker container ls - показывает работающие контейнеры

docker container ls –a – показывает все в том числе и остановленные

тоже самое

docker ps

docker ps -a

docker container inspect 2731223e570f - показывает настройки контейнера

docker container inspect 2731223e570f | grep IPAddress – позволяет отфильтровать вывод

docker exec -it b8e45bef8a83 bash - запускает дополнительный процесс в запущенном контейнере

docker run -d --name my\_nginx nginx - запуск кнтейнера с именем

docker run -d -p 8080:80 nginx — публикация портов внешний:контайнер

docker run -d -v ${PWD}:/usr/share/nginx/html -p 8080:80 nginx - ${PWD} -показывает путь до текущей папки

docker run -d -v /home/vladimir/Downloads/Repository/PetProjects/MyDoc/Doker/nginx:/usr/share/nginx/html -p 8081:80 nginx - абсолютный путь

запуск с разбиением

docker run \

--name my-nginx2 \

-v /home/vladimir/Downloads/Repository/PetProjects/MyDoc/Doker/nginx:/usr/share/nginx/html \

-p 8083:80 \

-d \

nginx

docker stop 2731223e570f - останавливает работу контейнера

docker kill 2731223e570f

docker container --rm имя или номер eудаляет контейнер после остановки

docker container prun –f - удаляет все остановленные контейнеры без переспрашивания

docker run –it **–d** Ubuntu bash – позволяеь запустить в фоновом режиме

docker attach имя контейнера позволят зайти в конкретный контейнер

для выхода Ctrl + p после Ctrl + q

docker run -it -d --name test2 --rm ubuntu bash – задает имя контейнера, удаляет после остановки

# IMAGE

docker history mongo – история создания образа

docker inspect mongo – информация об образе

docker save имя\_образа > transfer.tar - сохраняет образ в виде файла

docker load -i transfer.tar - загружает образ в систему

или

docker save –output nginx.tar nginx – сохраняет образ в виде файла

docker import nginx.tar svmyhome\nginx1 - загружает образ в систему

docker image pull ubuntu – скачать образ

docker image push svmyhome/catnip – запушить в репозиторий

docker image push my\_repo/my\_image:my\_tag – с тегом

docker images – список образов

docker image ls - – список образов

docker images --format "{{.Repository}} {{.Tag}}" – список нужных полей

docker images --format "table {{.Repository}} \t {{.Tag}}" – в виде таблицы лучшее читается

docker rmi hello-world – удаление образа

docker container prune — удаление всех остановленных контайнеров

docker image rm python:3-onbuild – тоже

docker image rm -f $(docker images -a -q) – удаление всех образов

docker image prune – удаляет образы none

docker tag svmyhomenginx1:latest 11111 – создает тег

## DIVE

dive id -утилита для анализа образа и его слоев

# DOCKERFILE

# - комментарий

ARG CODE\_VERSION=latest – аргумент сборки только на момент сборки

ARG MY\_ARG – можно передать в него значение во время сборки и использовать

FROM node:latest -из чего образ сделан

Label version=”1.0” -указывает версию

MAINTAINER Vladimir -кто автор образа

USER root -под кем будет запускаться

WORKDIR /opt/app – создется папка где будет запускаться, можно использовать несколько

ADD \*.json ./ - копирует все json\

ADD –-chown=root files\* /somedir

ADD my\_arc.tar.gz /somedir/ -разархивирует архив

ADD [http://url.tar.gz](http://url.tar.gz/) /somedir/

COPY . . - копирует из текущей директории в workdir

COPY \*.json ./ -только копирует

SHELL [“bin/sh”, ”-c”] – позволяет переопределить дефолтный басш

RUN echo Hello – запуск команды

ENV Foo=1 – определили переменную, остается в собранном образе

ENV Foo1=$Foo

VOLUME [“/data”] – создает volume на хост машине

ENTRYPOINT [‘top’,’-c’] – команды запуска

CMD [‘node’, “./cdsmkcmd/1.js”]

CMD [‘python’, “main.py ”]

STOPSIGNAL 9 сигнал при останвке контейнера

EXPOSE 80/TCP – Указывает на порт но не пробрасывает его



## build

docker build . -t testapi:latest -собирает образ, при сборке складывает все папки и подпапки

docker build -f ./apps/api/Dockerfile -t testapi:latest . собирает образ если докер находится в другом месте

docker build --no-cache -t test1:latest . Пересобирает образ по новой не используя кэш

## Multistaging build

Multistaging build– позволяет пере использовать то что создано в другом билде и уменьшить объем текущего билда за счет не включения ненужных сборок

FROM node:14-alpine as build

WORKDIR /opt/app

ADD \*.json ./

RUN npm install

ADD . .

RUN npm run build api

FROM node:14-alpine

WORKDIR /opt/app

ADD package.json .

RUN npm install --only=prod

COPY --from=build /opt/app/dist/apps/api ./dist

CMD ["node","./dist/main.js"]

# NETWORK

bridge – изолированная сеть между контейнерами (default)

host – та же сеть что и у хоста

overlay - docker Swarm

macVlan – на контейнер индивидульный мак выделяется

null – без сети

## Команды

docker network create my-net1 – создание сети

docker network connect my-net1 node2 – подключение к созданной сети

docker network disconnect my-net1 – отключение от сети

docker network inspect bridge– проверка конкретной сети (показывает в containers только запущенные контейнеры)

docker network ls – список сетей

docker network prune -удаление

docker network rm - удалени

docker run --name=node3 --network my-net1 -d demo3:latest

docker run -d --name=node4 -p 3000:3000 --network my-net1 demo3:latest

# MOUNT FILES AND DISKS

## VOLUMES

docker volume create NAME Создание volume

docker volume inspect NAME вывод информации о volumes

docker volume ls список volumes

docker volume prune удаление неиспользуемых volumes

docker volume rm NAME удаление одного volumes

docker run -d --name volume-3 -p 3003:3000 -v vol-1:/opt/app/data demo4:latest мы монтируем содерждимое хостового volume vol-1 в папку data образа :/opt/app/data и в дальнейшем пишем туда. При монтирование содержимое папки в образе замещается файлами из volume

полный синтаксис:

docker run -it --mount type=volume,src=v1,dst=/opt/app demo:latest

### volume in Dockerfile

FROM node:14-alpine

RUN mkdir postman1

WORKDIR /postman1

VOLUME [“/post”] - в докерфайле объявляется каталог содержимое которого будет биндится в volume на хост машине. При создании image не создается, создается только при создании контейнера:

docker run -it --name post3 -v demo-postman:latest – создаст на хост машине каталог вида: d1f0b6b90d786575b21b7594367d5a726222538b80b4

Чтобы создать удобный и читаемый Volume нужно create volume Имя

Далее дополнительно указать как в примере:

docker run -it --name post3 -v Имя:/postman1 demo-postman:latest

docker run --rm --mount type=volume,src=my\_python,target=/bind python python /bind/python.py

## BIND папки

docker run -it --name post3 -v /home/vladimir/docker-demo-4:/postman demo-postman:latest - монтирует конкретный каалог хостовой тачки с файлави в папку образа

полный синтаксис:

docker run -it --mount type=bind,source=/home/vladimir/docker-demo-4,target=/opt/app demo-postman:latest

docker run -it --rm --mount type=volume,src=my\_python,target=/bind python python /bash/python.py

## TRPFS

Записывается в память и полезно только для секретных данных

## Копирование файлов и папок

### В контейнер

docker cp /home/vladimir/temp2/1.txt b487a6211a48:/tmp – скопировать конкретный файл в докер

docker cp /home/vladimir/temp2/. b487a6211a48:/tmp – скопировать все содержимое папки temp2 в tmp

docker cp /home/vladimir/test1/ b487a6211a48:/tmp – скопировать папку test1 с содержимым в папку tmp

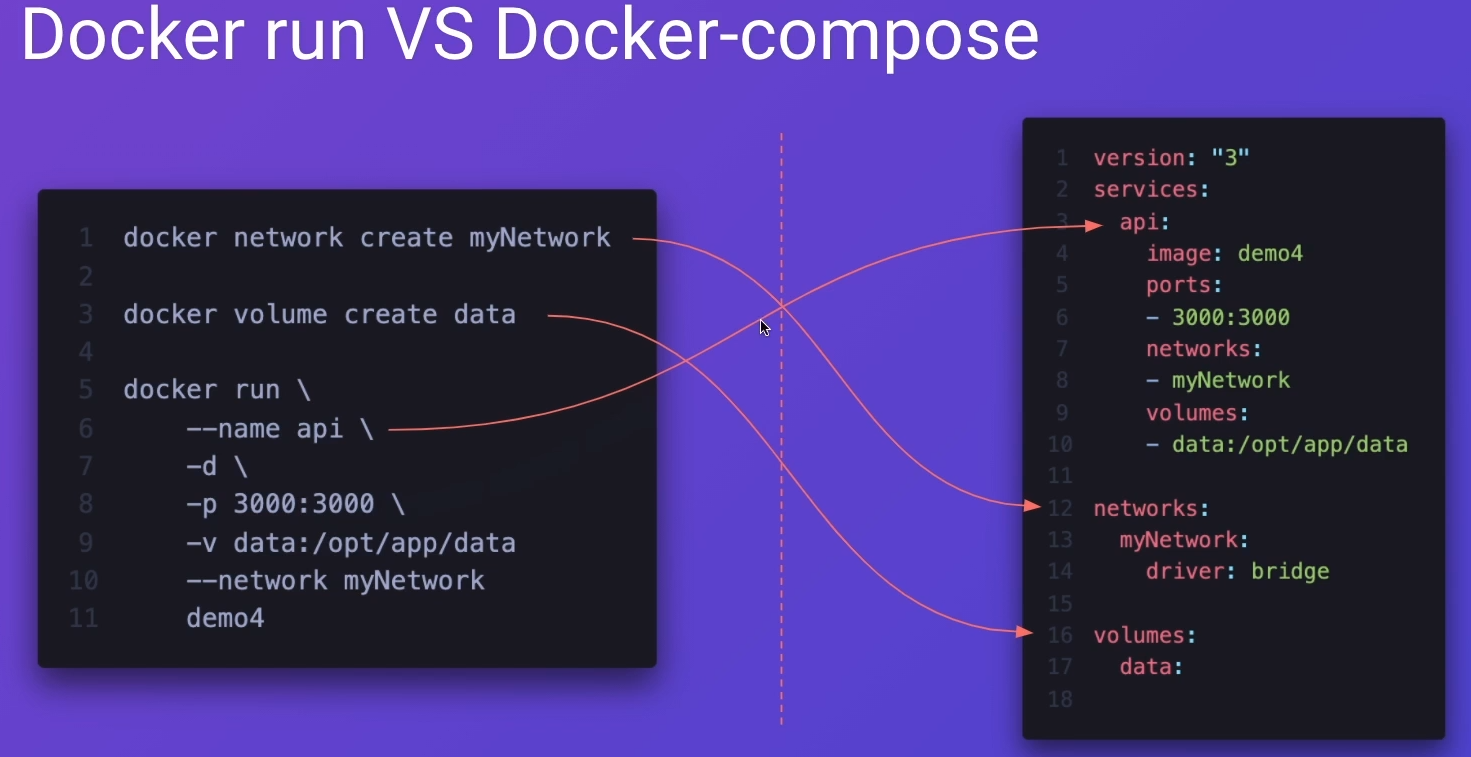
### Из контейнера

docker cp b487a6211a48:/tmp/1.txt /home/vladimir/1111 – скопироват конкретный файл из в докер

docker cp b487a6211a48:/tmp/. /home/vladimir/1111 – скопировать все содержимое папки tmp c подпапками и файлами

docker cp b487a6211a48:/tmp/test1 /home/vladimir/1111– скопировать папку test1 с содержимым из папки tmp

# DOCKER COMPOSE



## Install

<https://docs.docker.com/compose/install/>

## Команды

docker-compose --version

docker-compose up – поднимает все файлы в docker-compose

docker-compose stop – остановить контейнеры из списка docker-compose

docker-compose start – запустить контейнеры из списка docker-compose

docker-compose down – остановить контейнеры из списка docker-compose и удалить все неиспользуемое содержимое

docker-compose logs – логи по контайнерам

docker-compose images – список всех images используемых в docker-compose

docker-compose top – список запущенных процессов

docker-compose --profile start\_collection up -d - в docker-compose прописывается профиль и позволят запускать конкретные блоки

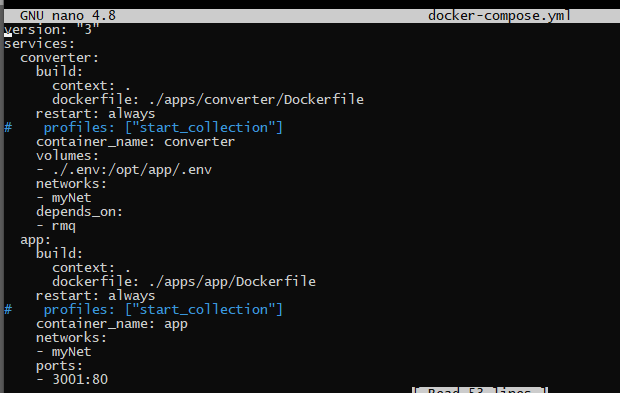
docker-compose run api – позволяет запустить один сервис

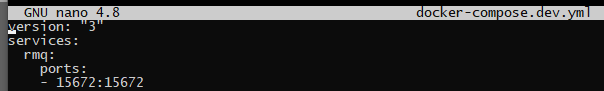
docker-compose --env-file .env.compose up -d - собирает переменные в docker-compose.yml из .env.compose Необходимо в docker-compose изменить переменную на container\_name: "${NAME}"

docker-compose --env-file .env.compose config - показывает как будет выглядеть итоговый файл с переменными из .env.compose

COMPOSE\_PROJECT\_NAME=11111 docker-compose --env-file .env.compose up -d - заменяет дефолтное имя проекты берущееся из названия папки на 11111

docker-compose -f docker-compose.yml -f docker-compose.dev.yml up -d - через 2-ой файл можно дополнить конфигурации сервиса, например портами



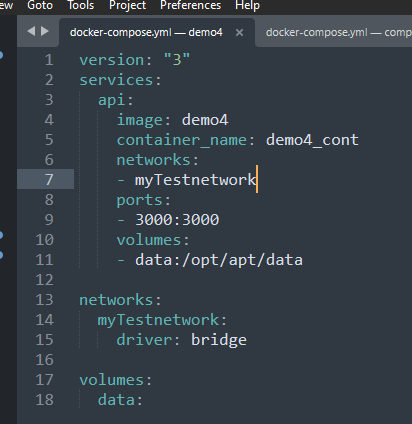


docker-compose -f docker-compose.yml -f docker-compose.dev.yml config – соберет и покажет такую конфигурацию на экране

docker-compose up – d 2-ой вариант наследования указывать в коде extends и дополнить еще один файл

|  |  |
| --- | --- |
|  |  |

## Примеры:



version: "3"

services:

api:

build: # собирает из докер файла

context: .

dockerfile: ./apps/api/Dockerfile

restart: always # перестартовывает сервис в случае отключения

container\_name: api

volumes:

- ./.env:/opt/app/.env

networks:

- myNet

depends\_on: # вначале запускается образ указанный в зависимости

- rmq

rmq:

image: rabbitmq:3-management

restart: always

networks:

- myNet

environment: # передает внутрь докера переменные

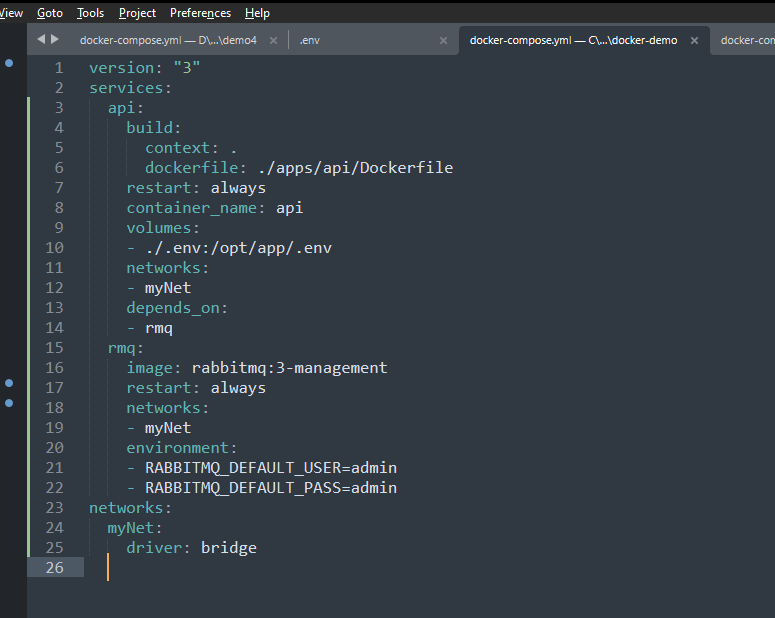
- RABBITMQ\_DEFAULT\_USER=admin

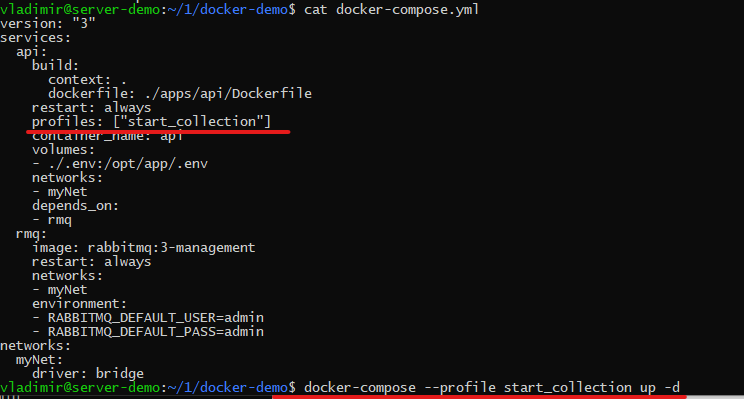
- RABBITMQ\_DEFAULT\_PASS=admin

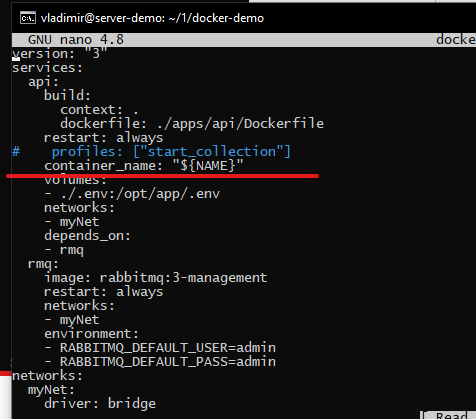
networks:

myNet:

driver: bridge







version: "3"

services:

api:

build:

context: .

dockerfile: ./apps/api/Dockerfile

restart: always

# profiles: ["start\_collection"]

container\_name: "${NAME}"

volumes:

- ./.env:/opt/app/.env

networks:

- myNet

depends\_on:

- rmq

rmq:

image: rabbitmq:3-management

restart: always

networks:

- myNet

environment:

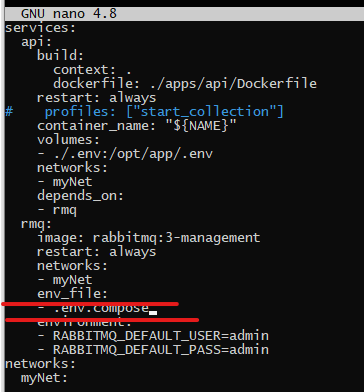
- RABBITMQ\_DEFAULT\_USER=admin

- RABBITMQ\_DEFAULT\_PASS=admin

networks:

myNet:

driver: bridge



GNU nano 4.8 docker-compose.yml Modified services:

api:

build:

context: .

dockerfile: ./apps/api/Dockerfile

restart: always

# profiles: ["start\_collection"]

container\_name: "${NAME}"

volumes:

- ./.env:/opt/app/.env

networks:

- myNet

depends\_on:

- rmq

rmq:

image: rabbitmq:3-management

restart: always

networks:

- myNet

env\_file:

- .env.compose

networks:

myNet: