

# Практическое применение docker

## часть 2

Евгений Мисяков  
SRE-инженер в Нетология



# Евгений Мисяков

О спикере:

- 15 лет в IT.
- Опыт работы DevOps/SRE 6 лет



# Docker полезности



1

# Сканер уязвимостей. Yandex container registry

default

Container Registry / Реестры / / ruby

Занято 1.43 ГБ

Обзор

Жизненный цикл

Права доступа

ruby

Docker-образы

Фильтр по хешу или тегу

Дата создания	Статус	Теги	Размер	Уязвимости	Статус сканирования	Дата последнего сканирования	
04.12.2023, в 14:33	Active	3.1.3-upg1	399.35 МБ	12 227 357 754	Готово	04.12.2023, в 14:34	...
04.12.2023, в 14:25	Active	3.1.3	338.65 МБ	21 334 495 778	Готово	04.12.2023, в 14:34	...

## Pull-команда

С помощью этой команды можно скачать Docker-образ из репозитория по тегу:

```
$ docker pull cr.yandex/[REDACTED]/ruby:3.1.3
```

# Docker capabilities.

`docker run --rm -it alpine sh`

```
yoga7% docker run --rm -it alpine sh
/ # ls /dev
console fd mqueue ptmx random stderr stdout urandom
core full null pts shm stdin tty zero
/ #
```

`docker run --rm --privileged -it alpine sh` - Привилегированный режим(все CAP)

```
See 'docker run --help'.
yoga7% docker run --rm --privileged -it alpine sh
/ # ls /dev
HID-SENSOR-2000e1.2.auto loop10 tty18 ttyS19
HID-SENSOR-2000e1.3.auto loop11 tty19 ttyS2
HID-SENSOR-2000e1.5.auto loop12 tty2 ttyS20
```

`docker run --cap-add <название CAP>` — разрешение на выполнение указанных capabilities

описание доступных capabilities: <https://www.opennet.ru/man.shtml?topic=capabilities&category=7&russian=0>

# SystemD для ansible molecule.

```
docker run -d --privileged --name test_systemd_centos oowy/centos:stream8
```

```
docker run -d --privileged --name test_systemd_ubuntu oowy/ubuntu:20.04
```

```
docker exec -it test_systemd_ubuntu bash
```

```
apt update && apt install -y curl
```

```
bash -c "$(curl -L https://setup.vector.dev)"
```

```
apt install -y vector
```

```
systemctl start vector && systemctl status vector
```

```
docker rm -f test_systemd_ubuntu
```

# Docker daemon remote connection

```
systemctl edit docker.service
```

```
[Service]
```

```
ExecStart=
```

```
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock  
-H tcp://127.0.0.1:2375
```

```
systemctl daemon-reload && systemctl restart docker
```

```
root@docker:~# ss -tlpn
```

State	Local Address:Port	Peer Address:Port	Process
LISTEN	*:2375	*:*	
users:(("dockerd",pid=4691,fd=3))			

# Docker-cli context

```
docker context create remote-ssh --docker host=ssh://<user>@<host>
```

```
docker context create remote-tcp --docker host=tcp://<host>:2375
```

```
docker context use remote-ssh
```

```
docker context ls
```

NAME	DESCRIPTION	DOCKER ENDPOINT
default	Current DOCKER_HOST based configuration	unix:///var/run/docker.sock
remote-ssh *		ssh://ubuntu@158.160.78.243
remote-tcp		tcp://158.160.78.243:2375

## Альтернатива:

```
export DOCKER_HOST="ssh://<user>@<host>"
```

```
<docker commands>
```



# Docker-cli context

```
docker context inspect remote-tcp
```

```
[
  {
    "Name": "remote-tcp",
    "Metadata": {},
    "Endpoints": {
      "docker": {
        "Host": "tcp://xxx.xxx.xxx.xxx:2375",
        "SkipTLSVerify": false
      }
    },
    "TLSMaterial": {},
    "Storage": {
      "MetadataPath":
"/home/user/.docker/contexts/meta/b4333411bc73f3109bc3b4fd06789936e59a55ba4f6d967c4efd35513e0f37a6",
      "TLSPath":
"/home/user/.docker/contexts/tls/b4333411bc73f3109bc3b4fd06789936e59a55ba4f6d967c4efd35513e0f37a6"
    }
  }
]
```

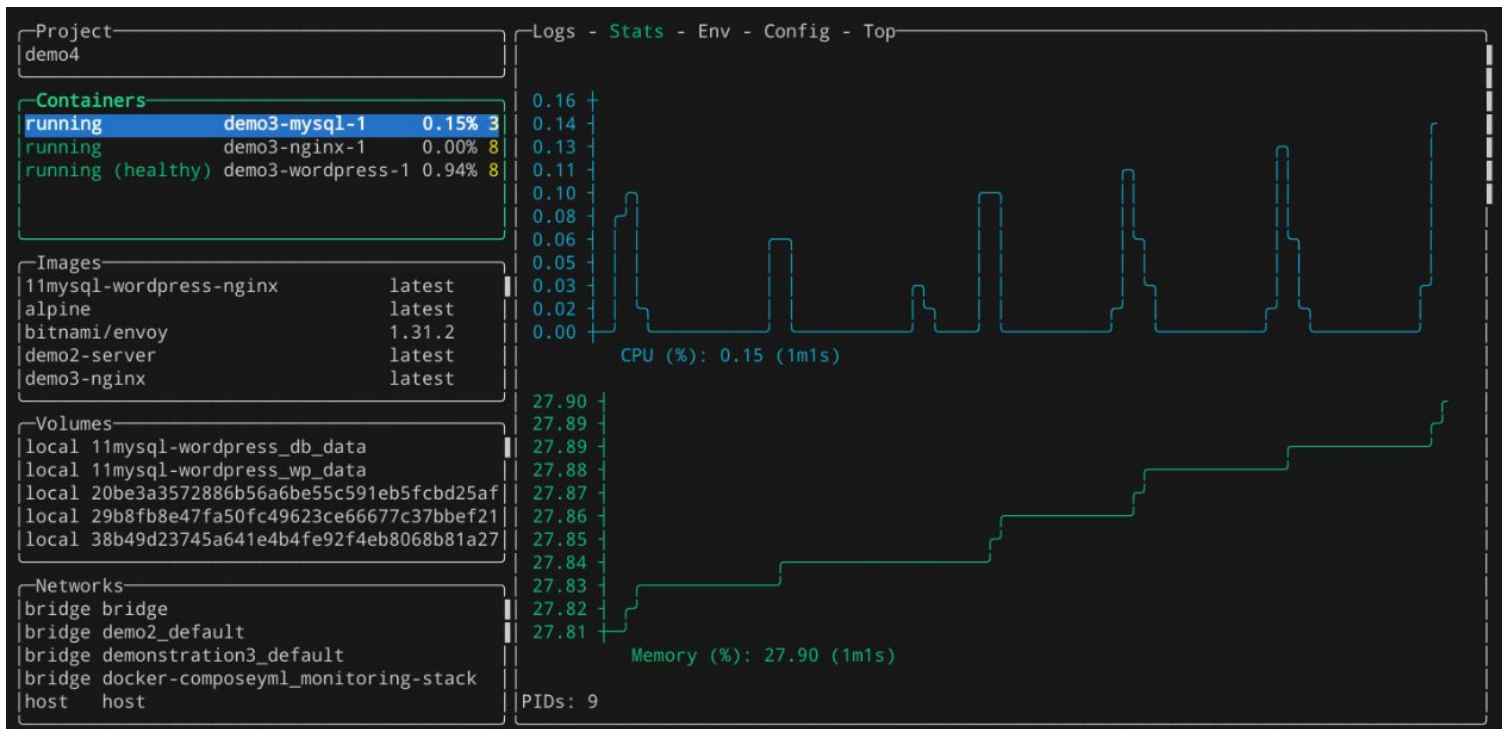
```
docker context export remote-tcp
```

```
Written file "remote-tcp.dockercontext"
```

# lazydocker

<https://github.com/jesseduffield/lazydocker#installation>

```
curl https://raw.githubusercontent.com/jesseduffield/lazydocker/master/scripts/install_update_linux.sh | bash
```



# Docker Compose Advanced



2

# Проблема мульти контейнерных конфигурации Docker

Команда **docker run** применяется для **императивного** управления контейнерами. Для достижения результата необходимо последовательно выполнять docker-cli команды.

Запуск вручную мульти контейнерных конфигурации при этом может выглядеть вот так:

```
#Создаем сеть 'wordpress'
```

```
docker network create --driver=bridge wordpress
```

```
#Запускаем контейнер с MySQL в сети 'wordpress'
```

```
docker run -d --network='wordpress' --hostname='db' -v 'db_data:/var/lib/mysql' -e  
'MYSQL_ROOT_PASSWORD=somewordpress' -e 'MYSQL_DATABASE=wordpress' -e 'MYSQL_USER=wordpress' -e  
'MYSQL_PASSWORD=wordpress' mariadb:10.6.4-focal --default-authentication-plugin='mysql_native_password'
```

```
#Запускаем контейнер с wordpress в сети 'wordpress'
```

```
docker run -d --network='wordpress' --hostname='wordpress' -v 'wp_data:/var/www/html' -p '80:80' -e  
'WORDPRESS_DB_HOST=db' -e 'WORDPRESS_DB_USER=wordpress' -e 'WORDPRESS_DB_PASSWORD=wordpress' -e  
'WORDPRESS_DB_NAME=wordpress' wordpress:latest
```

Тк метод не соблюдает принципы **идемпотентности** мы не будем терять время и вдаваться в тонкости работы с docker run



# docker-compose или docker compose?

## **docker info**

Client: Docker Engine - Community

Version: 24.0.7

Context: default

Debug Mode: false

Plugins:

buildx: Docker Buildx (Docker Inc.)

Version: v0.11.2

Path: /usr/libexec/docker/cli-plugins/docker-buildx

compose: Docker Compose (Docker Inc.)

Version: **v2.21.0**

Path: /usr/libexec/docker/cli-plugins/docker-compose

## **docker compose version**

Docker Compose version v2.21.0

## **docker-compose --version**

Docker Compose version v2.12.2

## **which docker-compose**

/usr/local/bin/docker-compose

# Пример простейшего docker-compose.yml для registry

```
version: '3'
services:
  registry:
    image: registry:2
    ports:
      - 5000:5000
    volumes:
      - registry_data:/data
volumes:
  registry_data: {}
```

**docker-compose up -d**

**docker-compose ps**

NAMES	COMMAND	SERVICE	STATUS	PORTS
<u>example-registry-1</u>	"/entrypoint.sh /etc..."	registry	running	0.0.0.0:5000->5000/tcp

# Render docker-compose config

## docker compose config

```
name: example_registry
services:
  registry:
    image: registry:2
    networks:
      default: null
    ports:
      - mode: ingress
        target: 5000
        published: "5000"
        protocol: tcp
>>>
```

```
>>>
  volumes:
    - type: volume
      source:
        registry_data
        target: /data
        volume: {}
  networks:
    default:
      name: example_default
  volumes:
    registry_data:
      name:
        example_registry_data
```

# Использование переменных окружения

```
version: '3'
services:
  db:
    image: mysql:${MYSQL_VERSION:-8}
    environment:
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD:-p@ssw0rd}'
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=${MYSQL_PASSWORD:-p@ssw0rd}
    env_file:
      - /opt/secrets/wordpress_env
  wordpress:
    image: wordpress
    environment:
      - WORDPRESS_DB_NAME=wordpress
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=${MYSQL_PASSWORD:-p@ssw0rd}
      - WORDPRESS_DB_HOST=mysql
    env_file:
      - /opt/secrets/wordpress_env
```

```
cat .env
MYSQL_ROOT_PASSWORD=rootpass
MYSQL_PASSWORD=wordpress
```

```
${VARIABLE:-default}
```



# Приоритет переменных окружения в Docker-compose

От высшего к низшему:

1. `docker compose run -e MYSQL_ROOT_PASSWORD=rootpass1`
2. `export MYSQL_ROOT_PASSWORD=rootpass2`
3. `environment: ["MYSQL_ROOT_PASSWORD=rootpass3"]`
4. `--env-file /opt/secrets/.env` (rootpass4)
5. `env_file: ["/opt/secrets/.env"]` (rootpass5)
6. `.env` file in project dir (rootpass5)
7. `Dockerfile ENV MYSQL_ROOT_PASSWORD=rootpass6`

# Резервы и лимиты

```
version: '3'
services:
  db:
    image: mysql
    deploy:
      resources:
        limits:
          cpus: "0.5"
          memory: 256M
        reservations:
          cpus: "0.25"
          memory: 128M
```

## **docker stats**

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %
58d66d772709	example-registry-1	0.00%	3.906MiB / 256MiB	1.53%

# Порядок запуска сервисов

```
version: '3'
services:
  db:
    image: mysql

  wordpress:
    image: wordpress
    depends_on: #Сервис ожидает успешного запуска сервиса db
      - db
```

Устаревший вариант: `links: ["db"]`. Метод также обеспечивал сетевую связанность, но в docker compose v3 все контейнеры, описанные в манифесте по-умолчанию находятся в одной сети

# Healthchecks

```
version: '3'
wordpress:
  image: wordpress
  healthcheck:
    test:
      [
        "CMD-SHELL",
        "curl -f http://127.0.0.1/wp-admin/install.php/ | grep 'Продолжить' || exit 1",
      ]
    interval: 10s
    timeout: 5s
    retries: 2
```

ТК БД в данном манифесте отсутствует - сервис выдаст статус unhealthy

# Restart policy

```
services:  
  db:  
    image: mysql  
    restart: on-failure
```

**no:** Контейнеры не перезапускаются автоматически

**on-failure[:max-retries]:** Перезапускает контейнер, если он завершает работу с ненулевым кодом выхода(те ошибка), можно указать максимальное количество попыток.

**always:** Контейнеры перезапускаются в случае завершения работы(**значение по умолчанию**)

**unless-stopped:** Всегда перезапускает контейнер, если только он не был остановлен пользователем или Docker-daemon.

# Использование yaml-anchor

```
version: '3'
x-deploy: &deploy-dev
  deploy:
    resources:
      limits:
        memory: 256M
x-env_file: &env_file
env_file:
  - .env

services:
  db:
    image: mysql
    <<: [*deploy-dev, *env_file]

  wordpress:
    image: wordpress
    <<: [*deploy-dev, *env_file]
```

Результат рендера. docker compose config

```
name: example
services:
  db:
    deploy:
      resources:
        limits:
          memory: "268435456"
    environment:
      MYSQL_PASSWORD: wordpress
      MYSQL_ROOT_PASSWORD: somewordpress
    image: mysql
  wordpress:
    deploy:
      resources:
        limits:
          memory: "268435456"
    environment:
      MYSQL_PASSWORD: wordpress
      MYSQL_ROOT_PASSWORD: somewordpress
    image: wordpress

x-deploy:
  deploy:
    resources:
      limits:
        memory: 256M
x-env_file:
  env_file:
  - .env
```

# Multiple compose files. Merge

```
#!/opt/wordpress/compose.yaml
```

```
version: '3'
```

```
services:
```

```
wordpress:
```

```
  image: wordpress
```

```
db:
```

```
  image: mysql
```

```
  command: --default-authentication-plugin=caching_sha2_password
```

```
  ports:
```

```
    - "127.0.0.1:3306:3306"
```

```
    - "192.168.99.1:3306:3306"
```

```
proxy:
```

```
  image: nginx
```

```
#!/opt/wordpress/compose.override.yaml
```

```
файл по-умолчанию
```

```
version: '3'
```

```
services:
```

```
db:
```

```
  command: --default-authentication-plugin=mysql_native_password
```

```
  ports:
```

```
    - "192.168.99.1:3306:3306"
```

```
proxy:
```

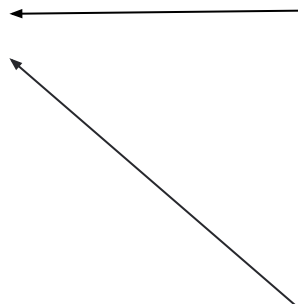
```
  image: nginx
```

docker compose up -f docker-compose.yaml -f docker-compose.prod.yaml -d

# Multiple compose files. Include

```
#/opt/wordpress/compose.yaml
version: '3'
include:
  - /opt/db/docker-compose.yaml
  - /opt/proxy/docker-compose.yaml
services:
  wordpress:
    image: wordpress
```

```
#/opt/db/compose.yaml
version: '3'
services:
  db:
    image: mysql
```

A diagram consisting of two arrows. One arrow originates from the 'include' section of the main compose file and points to the top-right box. The other arrow originates from the 'services' section of the main compose file and points to the bottom-right box.

```
#/opt/proxy/compose.yaml
version: '3'
services:
  proxy:
    image: nginx
```



# Docker Networks



3

# docker network drivers

## docker network ls

NETWORK ID	NAME	DRIVER
164293d59519	bridge	<b>bridge</b>

Сетевой драйвер по-умолчанию. Сетевой трафик между хостом и контейнером проходит через сетевой интерфейс docker0: inet 172.17.0.1/16 brd 172.17.255.255

147e1d77dba0	host	<b>host</b>
--------------	------	-------------

Прямое подключение к сетевому интерфейсу ОС. Не уменьшает производительность сети.

bc94d8ec6a80	legacy	<b>macvlan</b>
--------------	--------	----------------

Прямое подключение к физическому сетевому интерфейсу. Позволяет назначить Mac -адрес. Используется для легаси-приложений.

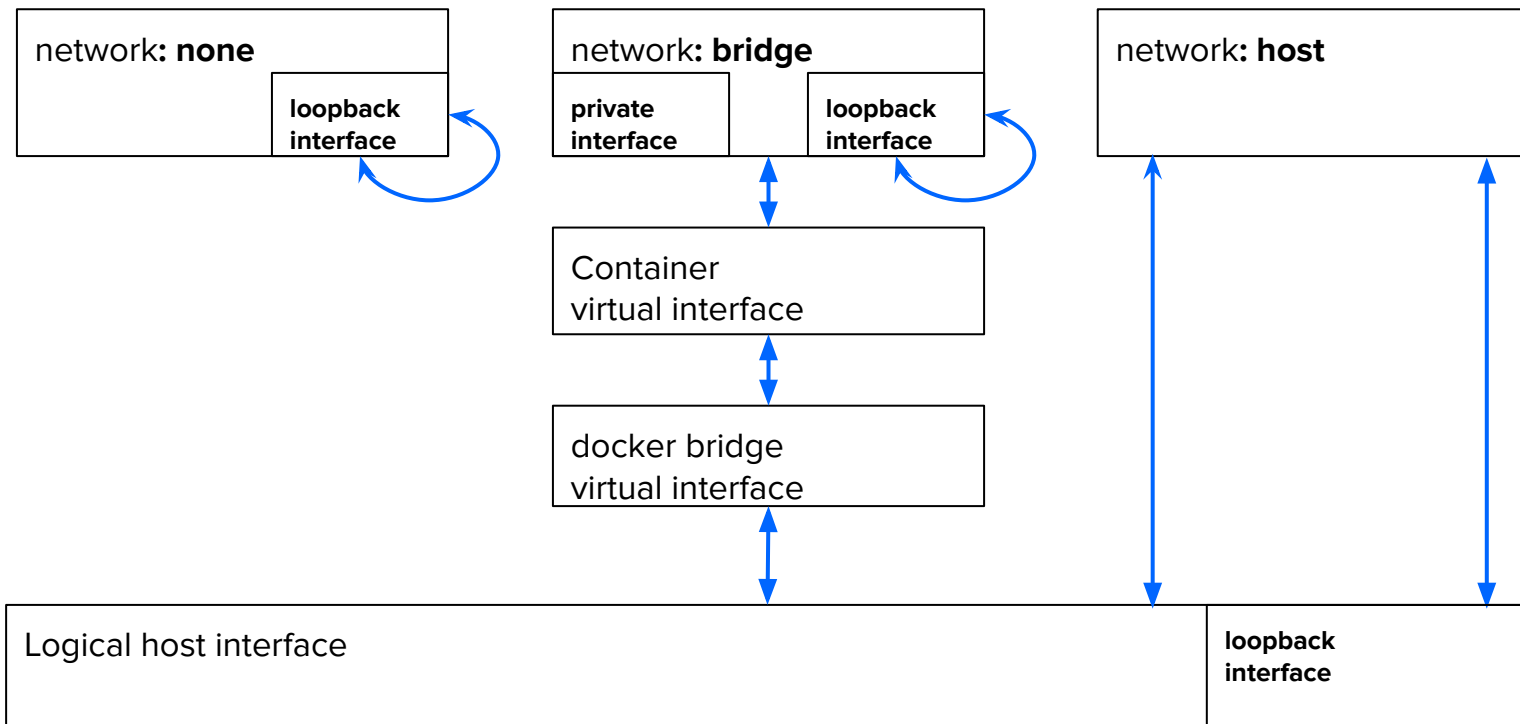
095515b8e08e	vlan	<b>ipvlan</b>
--------------	------	---------------

Позволяет создавать виртуальные интерфейсы в режимах I2/L3. Тегирует трафик VLAN ID

bc2709291a4e	none	<b>null</b>
--------------	------	-------------

--network none . Сетевая изоляция. Доступен только lo: 127.0.0.1 . Маппинг портов не работает!!

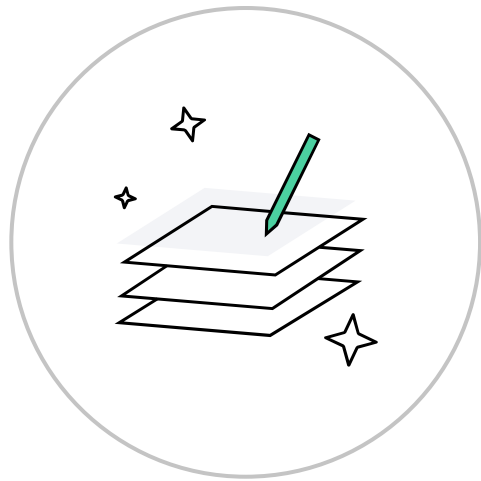
# Типы docker сетей



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#)

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



# Задавайте вопросы и пишите отзыв о лекции

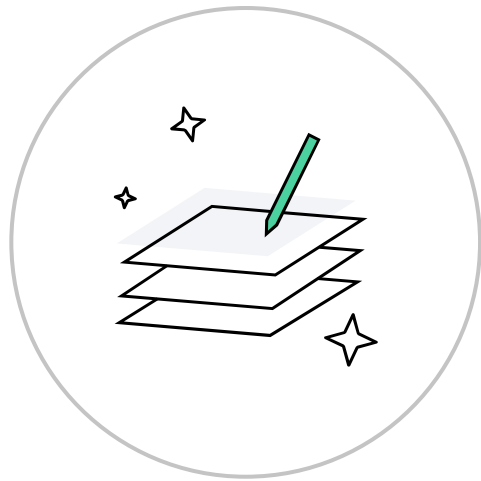
Евгений Мисяков  
SRE инженер в Нетологии



# Домашнее задание

Давайте посмотрим ваше домашнее задание.

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



# Дополнительные материалы

- История появления и развития контейнеров:  
<https://habr.com/ru/companies/serverspace/articles/741874/>
- Глубокое погружение в Linux namespaces:  
<https://habr.com/ru/articles/458462/>
- Cgroups:  
<https://habr.com/ru/companies/selectel/articles/303190/>
- Capabilities:  
<https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-privileged>
- Сборник обо всем:  
<https://awesome-docker.netlify.app/>

