

# Практическое применение docker

Евгений Мисяков  
SRE-инженер в Нетология



# Евгений Мисяков

О спикере:

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



# Docker “под капотом”



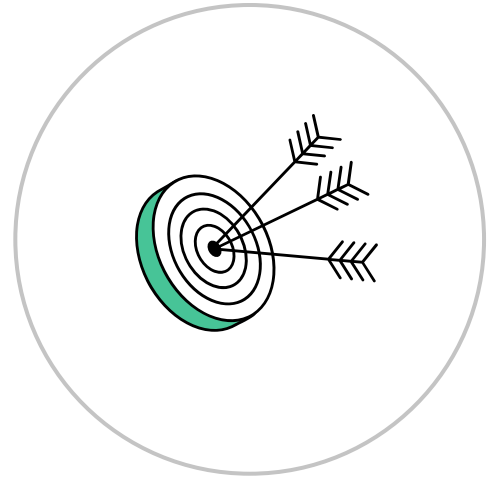
1

# Типичные задачи docker в 202X

- Песочница: (develop, test, debug, demo, **educate**)
- Запуск докеризированных приложений
- dind CI/CD pipelines
- Сборка образов контейнеров
- remote docker context
- тема для собеседования (\*\_\*)

# Цели занятия

- Обозначить роль Docker в современной разработке
- Освоить продвинутые методы сборки Dockerfile
- *“Погрузиться”* в docker-compose



# Механизмы контейнеризации docker



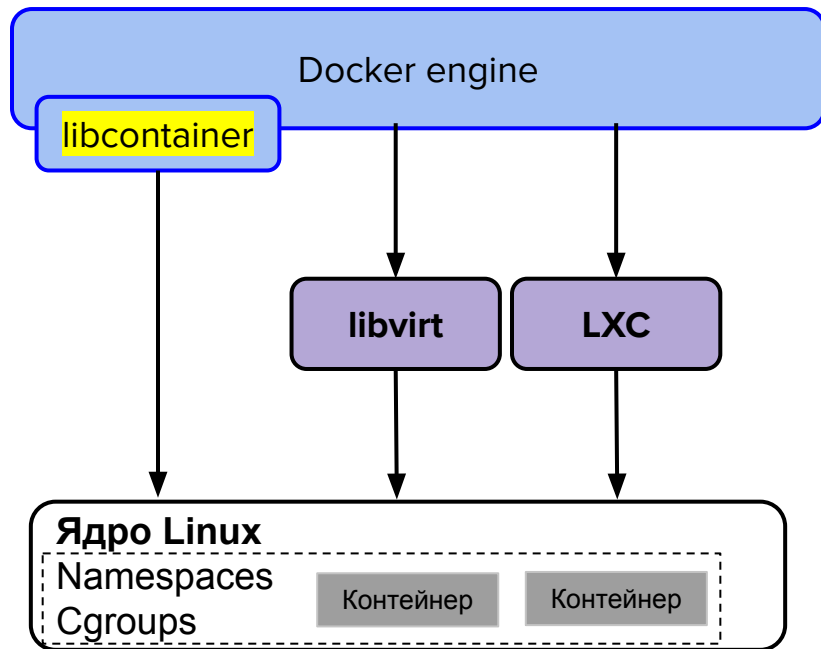
1

# История появления Docker

Дата	Событие
2013г. Docker	Инструмент для автоматизации развертывания контейнеров от dotCloud. <b>Упростил использование контейнеров</b>
2008г. LXC	Подсистема контейнеризации с использованием cgroups и изоляции на основе namespaces (которые постепенно появлялись в ядре linux с 2002 по 2013)
2006г. Process Containers	Контейнеризация от Google. Позволяла ограничить использование системных ресурсов. В 2007 Технология включена в ядро Linux как control groups (cgroups)
2005г. OpenVZ	Контейнеризация от Parallels, на основе пропатченного ядра linux
2004г. Oracle Solaris Containers	Первые в истории контейнеры на уровне операционной системы
2000г. FreeBSD Jails	Запускает процесс в <b>изолированной области</b> с сильно урезанными правами
1979г. Unix V7 chroot (change root)	<b>Подмена root-каталога</b> для процесса и его дочерних элементов. Процесс не может получить доступ к файлам вне этого корневого каталога

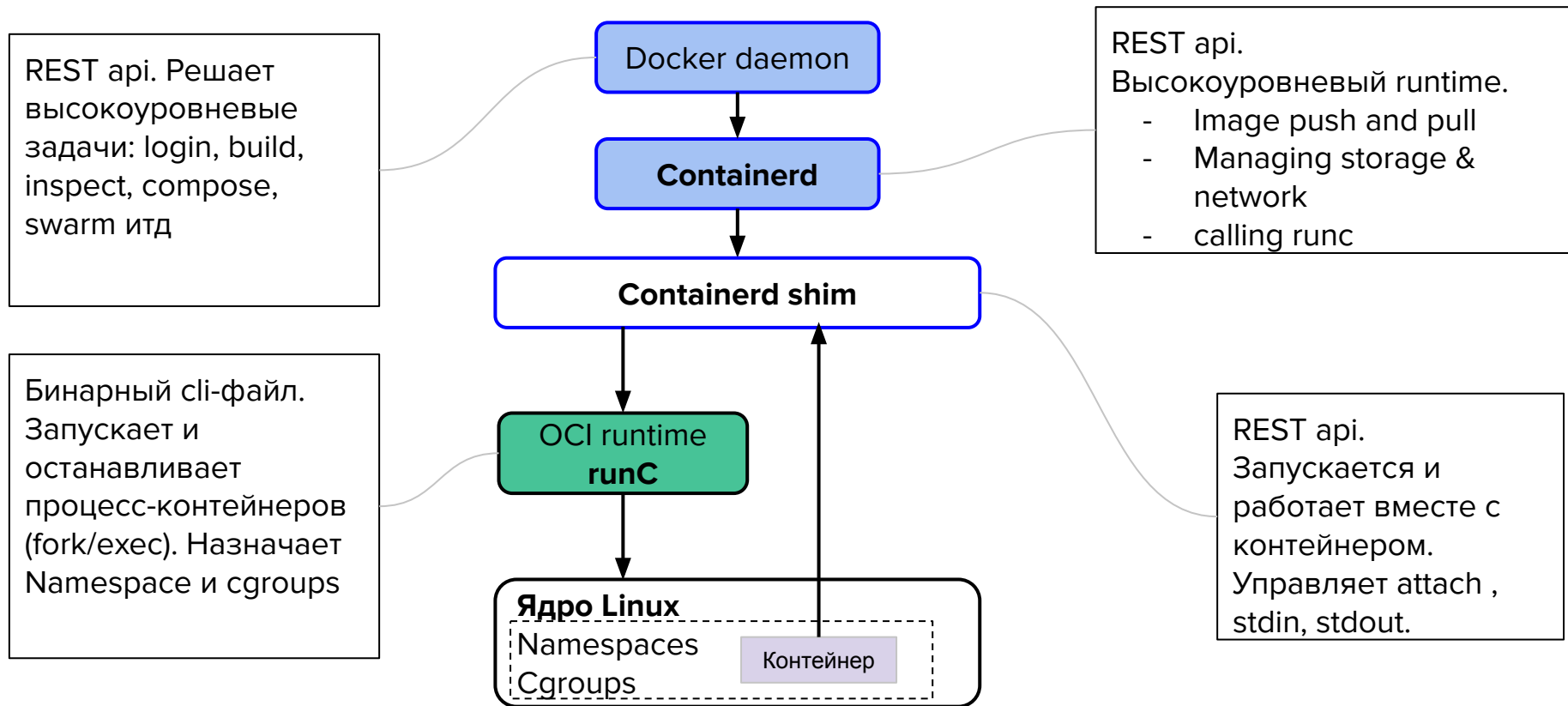
история появления контейнеризации - [ссылка](#).

# Устаревшие версии docker < 1.12.0





# Docker 1.12+ . OCI - определяет стандарт образов и контейнеров



# Использование Containerd

```
ctr images pull docker.io/library/nginx:latest
ctr run --detach --net-host docker.io/library/nginx:latest test-nginx
ctr container ls
```

CONTAINER	IMAGE	RUNTIME
test-nginx	docker.io/library/nginx:latest	io.containerd.runc.v2

```
ctr task ls
```

TASK	PID	STATUS
test-nginx	119630	RUNNING

```
ctr task kill test-nginx
```

TASK	PID	STATUS
test-nginx	119630	STOPPED

```
ctr container remove test-nginx
```

# Использование runc.

```
mkdir test_runc && cd test_runc
```

```
runc spec && ls  
config.json
```

```
mkdir rootfs  
apt-get -y install skopeo  
skopeo copy docker://docker.io/library/nginx:latest oci:nginx:latest
```

```
apt install umoci  
umoci unpack --image nginx:latest bundle  
cp -r bundle/rootfs config.json ./nginx && cd nginx
```

```
sed -i 's/terminal": true/terminal": false/' config.json  
sed -i 's/"sh"/"sleep","60"/' config.json
```

```
runc run nginx --detach
```

```
runc list
```

ID	ID	STATUS	BUNDLE	CREATED	OWNER
nginx	124495	running	/root/test_runc/nginx	11:49	root

```
runc kill nginx && runc delete nginx
```

<https://selectel.ru/blog/upravlenie-kontejnerami-s-runc/>

```
→ nginx tree -L 2
```

- blobs
  - sha256
- config.json
- index.json
- oci-layout
- rootfs
  - bin -> usr/bin
  - boot
  - dev
  - docker-entrypoint.d
  - docker-entrypoint.sh
  - etc
  - home
  - lib -> usr/lib
  - lib32 -> usr/lib32
  - lib64 -> usr/lib64
  - libx32 -> usr/libx32
  - media
  - mnt
  - opt
  - proc
  - root
  - run
  - sbin -> usr/sbin
  - srv
  - sys
  - tmp
  - usr
  - var

# containerd-shim

**docker run -d nginx**

46693e02c402a48d61adabe30b0c824cb6426f696aa4e5306df46741240a0317

**ps aux | grep shim**

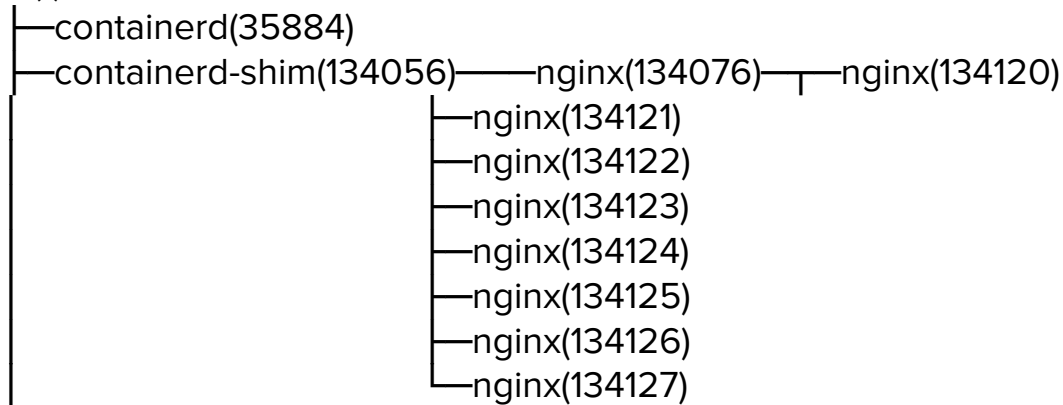
root 134056 /usr/bin/containerd-shim-runc-v2 -namespace moby -id

46693e02c402a48d61adabe30b0c824cb6426f696aa4e5306df46741240a0317 -address


/run/containerd/containerd.sock




**pstree -lpTs**


systemd(1)




# 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

<input type="checkbox"/> Дата создания	Статус	Теги	Размер	Уязвимости	Статус сканирования	Дата последнего сканирования	
<input type="checkbox"/> 04.12.2023, в 14:33	Active	3.1.3-upg1 	399.35 МБ	<div>12</div> <div>227</div> <div>357</div> <div>754</div>	Готово	04.12.2023, в 14:34	...
<input type="checkbox"/> 04.12.2023, в 14:25	Active	3.1.3 	338.65 МБ	<div>21</div> <div>334</div> <div>495</div> <div>778</div>	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
```

# Advanced dockerfile



2

# Инструкции Dockerfile. Кеширование слоев

**Dockerfile** – набор инструкций, описывающих сборку образа контейнера.

**Каждая** инструкция (кроме **FROM**) порождает новый слой в файловой системе. Лимит слоев 127 шт(и это явно намекает не плодить сущности без потребности!).

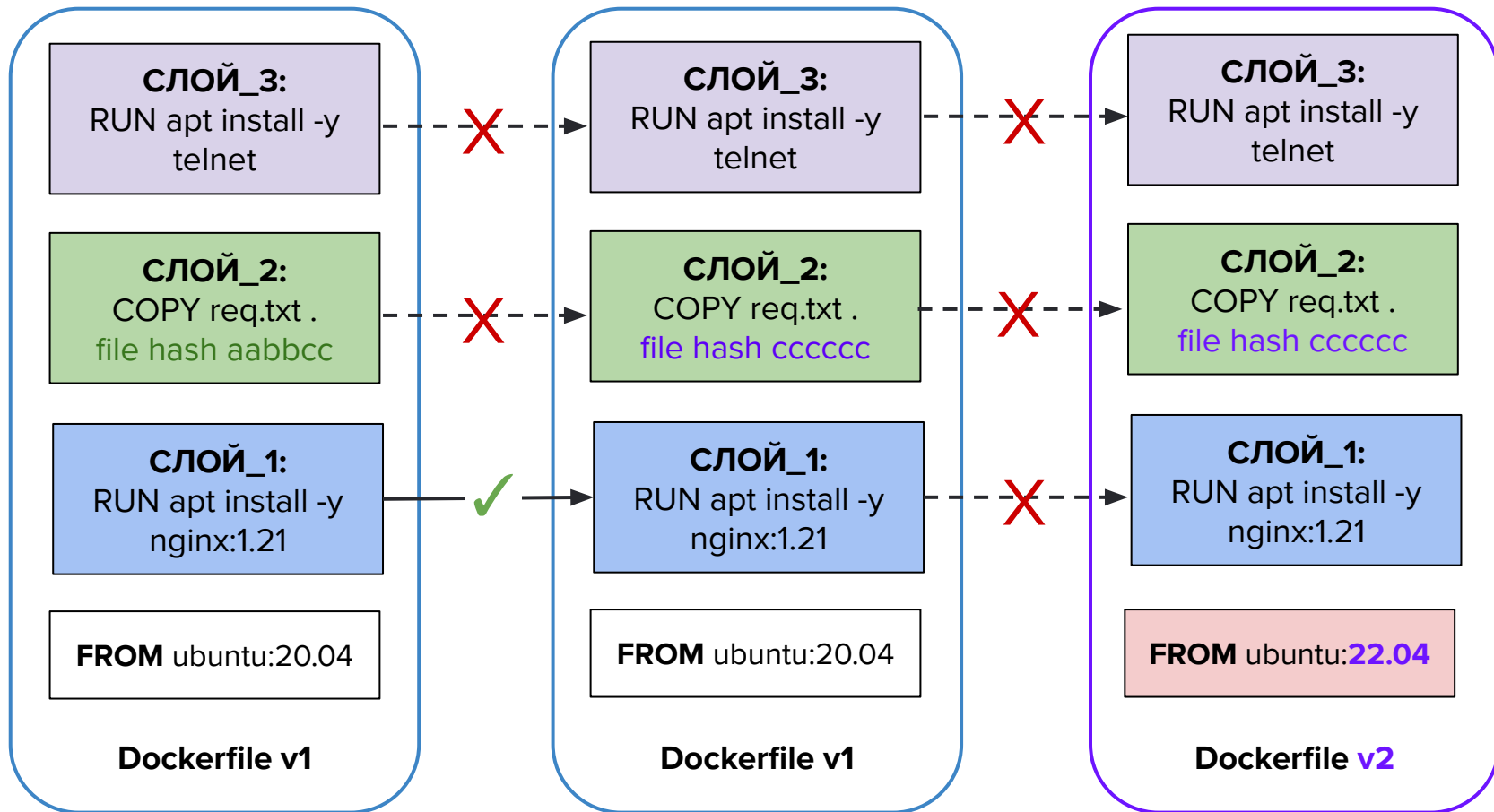
**ADD, COPY, RUN** – записывают все изменения в **новый слой**.

Остальные инструкции создают **пустой слой** и не влияют на размер образа.

При каждой новой сборке docker builder проверяет возможность переиспользования ранее сохраненных слоев (отключается ключом --no-cache).

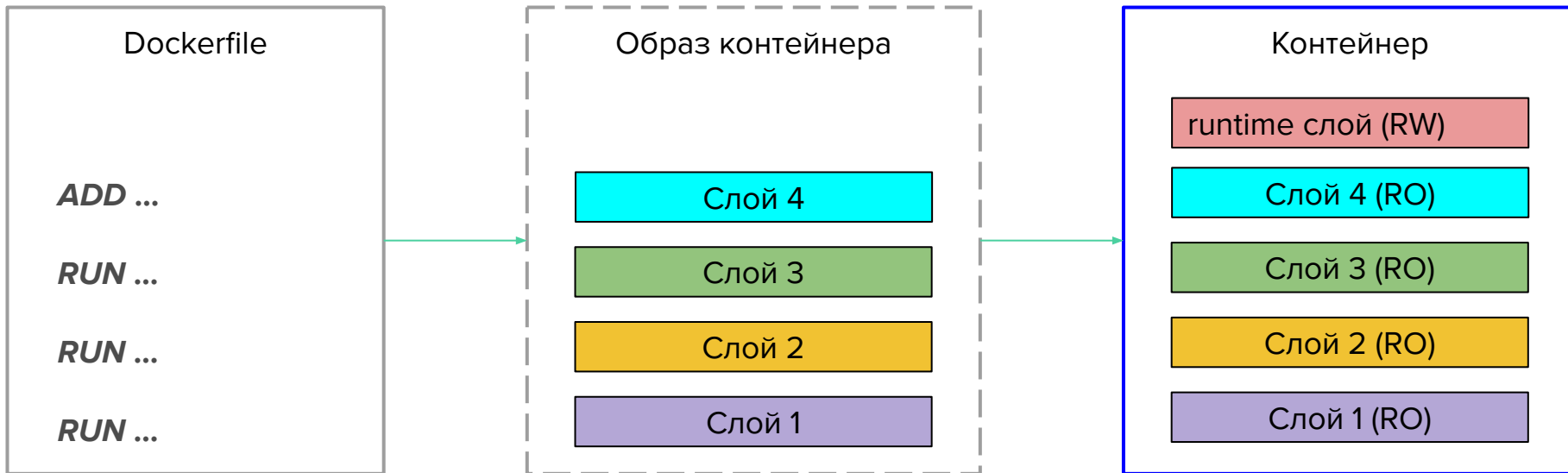
Правильный порядок инструкций влияет на эффективность использования кеша

# Dockerfile. Кеширование слоев





# Single-stage build или одноэтапная сборка.





## Вопрос

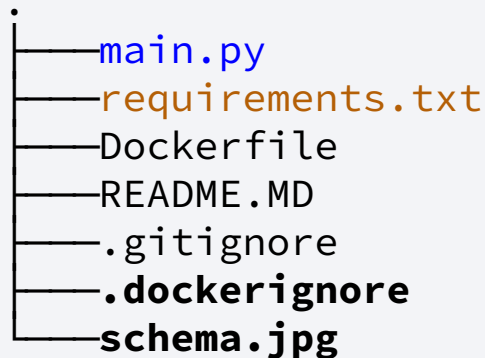
Здесь представлен правильный порядок инструкций?

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt main.py .
RUN pip install -r requirements.txt
CMD ["python", "main.py"]
```

```
.
├── main.py
├── requirements.txt
├── Dockerfile
├── README.MD
├── .gitignore
├── .dockerignore
└── schema.jpg
```

# Правильный порядок инструкций

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY main.py .
CMD ["python", "main.py"]
```



A diagram showing a directory structure. A vertical line on the left represents the root directory. From this line, horizontal lines branch out to the right, connecting to the following items: `main.py` (in blue), `requirements.txt` (in orange), `Dockerfile` (in black), `README.MD` (in black), `.gitignore` (in black), `.dockerignore` (in bold black), and `schema.jpg` (in blue).

**.dockerignore** помогает избежать копирования лишних файлов при использовании инструкции вида: **COPY ..**

# Docker inspect

```
docker inspect python_good_order | jq
```

```
    },
    "Name": "overlay2"
  },
  "RootFS": {
    "Type": "layers",
    "Layers": [
      "sha256:ec983b16636050e69677eb81537e955ab927757c23aaf73971ecf5f71fcc262a",
      "sha256:87579ac672ec5928830c3d44e850f826a545df026931d3436209a8858602aec7",
      "sha256:abc4ad0d31b0257e9111dedd49eba03cbaf3fb882511dd2f13c56af56d3e3d95",
      "sha256:3e71e1bda445b037dad9cefd943593fd14c235eff6fd16ce6e0fa9ec09553cbe",
      "sha256:d2ae857413fcae35dc6a6fab12178e01173a65e7b8fe89b3119aeeef7ab18329f",
      "sha256:b1d54ce976890c3ed1541976248716dd2839fbaea9362a44f68355470e9706b5",
      "sha256:46a589056c0a71d96a24b2355df7b008435a2bee929e3eb7758a4d9cab9adca0",
      "sha256:236d2fde2667d7ff9b68ead308a843ba422c466871d5f7f08dbac6aa101a9466",
      "sha256:999242cb84a0e6a1a8f8d0d073e477c8782bdcd01cfbb15b73b50ec229f01389"
    ]
  },
  "Metadata": {
    "LastTagTime": "2023-11-14T17:11:23.591605559+03:00"
  }
}
```

9 слоев

# Docker inspect --format

```
docker inspect --format '{{range .RootFS.Layers}}{{printf "%s\n"  
.}}}{{end}}' python_good_order
```

```
sha256:ec983b16636050e69677eb81537e955ab927757c23aaf73971ecf5f71fcc262a  
sha256:87579ac672ec5928830c3d44e850f826a545df026931d3436209a8858602aec7  
sha256:abc4ad0d31b0257e9111dedd49eba03cbaf3fb882511dd2f13c56af56d3e3d95  
sha256:3e71e1bda445b037dad9cefd943593fd14c235eff6fd16ce6e0fa9ec09553cbe  
sha256:d2ae857413fcae35dc6a6fab12178e01173a65e7b8fe89b3119aeeef7ab18329f  
sha256:b1d54ce976890c3ed1541976248716dd2839fbaea9362a44f68355470e9706b5  
sha256:46a589056c0a71d96a24b2355df7b008435a2bee929e3eb7758a4d9cab9adca0  
sha256:236d2fde2667d7ff9b68ead308a843ba422c466871d5f7f08dbac6aa101a9466  
sha256:999242cb84a0e6a1a8f8d0d073e477c8782bdcd01cfbb15b73b50ec229f01389
```

9 слоев



**Почему количество  
слоев не совпало с  
количеством  
инструкций?**

# Docker history

```
docker history python_good_order
```

```
yoga7% docker history python_good_order
```

IMAGE	CREATED	CREATED BY	SIZE	COMMENT
e3270b6af7d9	6 minutes ago	CMD ["python" "main.py"]	0B	buildkit.dockerfile.v0
<missing>	6 minutes ago	COPY main.py ./ # buildkit	252B	buildkit.dockerfile.v0
<missing>	7 minutes ago	RUN /bin/sh -c pip install -r requirements.t...	11.3MB	buildkit.dockerfile.v0
<missing>	8 minutes ago	COPY requirements.txt ./ # buildkit	5B	buildkit.dockerfile.v0
<missing>	19 minutes ago	WORKDIR /app	0B	buildkit.dockerfile.v0
<missing>	4 weeks ago	CMD ["python3"]	0B	buildkit.dockerfile.v0
<missing>	4 weeks ago	RUN /bin/sh -c set -eux; savedAptMark="\$(a...	11.1MB	buildkit.dockerfile.v0
<missing>	4 weeks ago	ENV PYTHON_GET_PIP_SHA256=22b849a10f86f5ddf7...	0B	buildkit.dockerfile.v0
<missing>	4 weeks ago	ENV PYTHON_GET_PIP_URL=https://github.com/py...	0B	buildkit.dockerfile.v0
<missing>	4 weeks ago	ENV PYTHON_SETUPTOOLS_VERSION=58.1.0	0B	buildkit.dockerfile.v0
<missing>	4 weeks ago	ENV PYTHON_PIP_VERSION=23.0.1	0B	buildkit.dockerfile.v0
<missing>	4 weeks ago	RUN /bin/sh -c set -eux; for src in idle3 p...	32B	buildkit.dockerfile.v0
<missing>	4 weeks ago	RUN /bin/sh -c set -eux; savedAptMark="\$(a...	31.3MB	buildkit.dockerfile.v0
<missing>	4 weeks ago	ENV PYTHON_VERSION=3.9.18	0B	buildkit.dockerfile.v0
<missing>	4 weeks ago	ENV GPG_KEY=E3FF2839C048B25C084DEBE9B26995E3...	0B	buildkit.dockerfile.v0
<missing>	4 weeks ago	RUN /bin/sh -c set -eux; apt-get update; a...	9.24MB	buildkit.dockerfile.v0
<missing>	4 weeks ago	ENV LANG=C.UTF-8	0B	buildkit.dockerfile.v0
<missing>	4 weeks ago	ENV PATH=/usr/local/bin:/usr/local/sbin:/usr...	0B	buildkit.dockerfile.v0
<missing>	13 days ago	/bin/sh -c #(nop) CMD ["bash"]	0B	
<missing>	13 days ago	/bin/sh -c #(nop) ADD file:fb8521c24ed75802...	74.8MB	

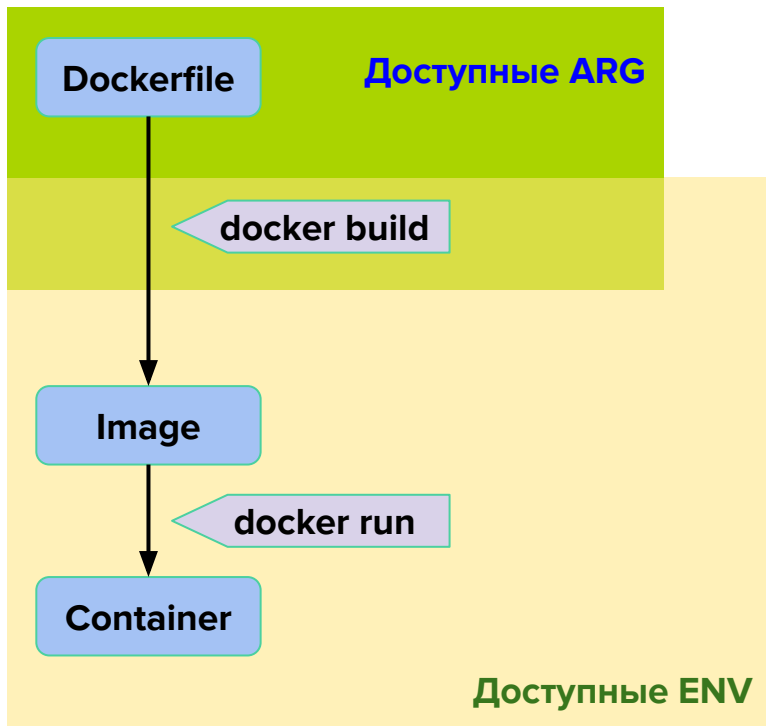
# hadolint

```
docker run --rm -i hadolint/hadolint < Dockerfile
```

```
-:4 DL3042 warning: Avoid use of cache directory with pip.  
Use `pip install --no-cache-dir <package>`
```



# Инструкции ENV и ARG



## Dockerfile:

#значение **ARG** по-умолчанию

**ARG** VERSION=latest FROM ubuntu:\${VERSION}

#значение **ENV** по-умолчанию

**ENV** VERSION=latest FROM ubuntu:\${VERSION}

# **ENV=ARG**

**ARG** PASSWORD

**ENV** PASSWORD=\${VERSION}

#Переопределение **ARG** при сборке

docker build --build-arg PASSWORD="qwerty"

#Переопределение **ENV** при запуске

docker run --env-file=./file\_name

docker run -e PASSWORD=P@55w0rD

# Инструкции ENV и ARG. Утечка секретов!

**docker history :**

**ENTRYPOINT** ["sleep" "infinity"]

**RUN** |2 ARG\_PASSWORD=P@55w0rD ARG\_DEFAULT\_PASSWORD=12345 /bin/sh -c  
echo \${ARG\_PASSWORD} > credentials.txt

**ENV** ENV\_ARG\_PASSWORD=P@55w0rD

**ENV** ENV\_DEFAULT\_PASSWORD=qwerty

**ARG** ARG\_DEFAULT\_PASSWORD=12345

**ARG** ARG\_PASSWORD

не делайте так !!!

Используйте **MOUNT secrets** или **MULTISTAGE BUILD**!

# Mount secrets

```
docker build --secret id=aws,src=$HOME/.aws/credentials .
```

## Dockerfile:

```
RUN apt install aws-cli
```

```
RUN --mount=type=secret,id=aws,target=/root/.aws/credentials aws s3 cp  
s3://bucket-name/db.sql db.sql
```

-----

```
eval $(ssh-agent) && ssh-add
```

```
docker build --ssh default=$SSH_AUTH_SOCK .
```

## Dockerfile:

```
RUN mkdir -p -m 0700 ~/.ssh && ssh-keyscan gitlab.com >> ~/.ssh/known_hosts
```

```
RUN --mount=type=ssh \  
ssh -q -T git@gitlab.com 2>&1 | tee /hello
```

-----

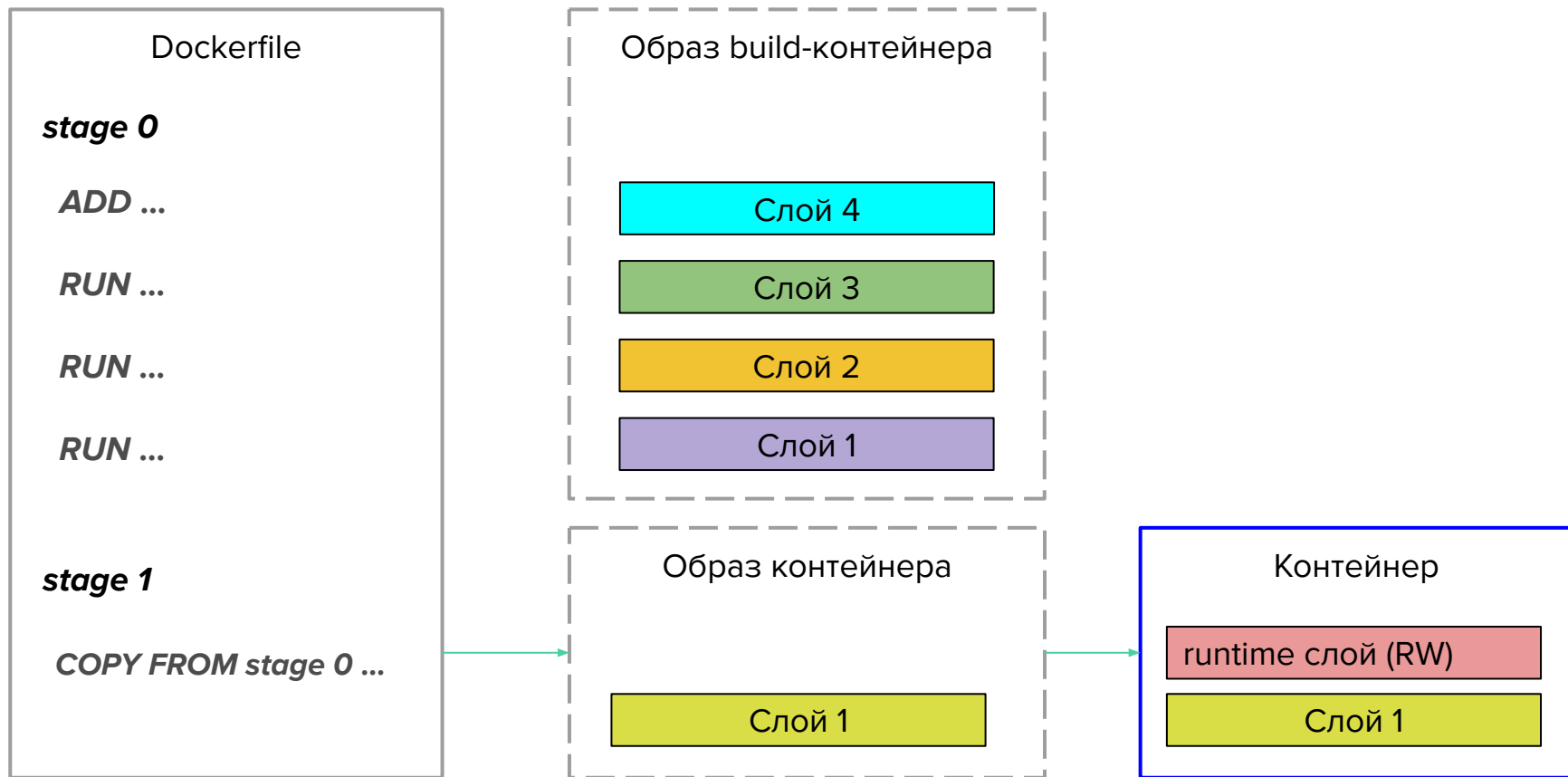
```
export MYSECRET=qwerty
```

```
docker build --secret id=env_sec,env=MYSECRET .
```

## Dockerfile:

```
RUN --mount=type=secret,id=env_sec < command > $(cat /run/secrets/MYSECRET)
```

# Multi-stage build или многоэтапная сборка



# Dive, docker save

```
docker run --rm -it \  
-v /var/run/docker.sock:/var/run/docker.sock \  
wagoodman/dive:latest python_good_order
```

<https://github.com/wagoodman/dive>

The screenshot displays the Docker Dive application interface. The left sidebar shows the 'Layers' section with a list of image layers, their sizes, and commands. The main panel is divided into two sections: 'Current Layer Contents' and 'Filetree'. The 'Current Layer Contents' section shows a list of files and their sizes for the selected layer. The 'Filetree' section shows a hierarchical view of the file system, including directories like 'app', 'bin', 'boot', 'dev', 'etc', 'pwd.lock', 'alternatives', 'README', 'awk', 'builtins', 'nawk', 'pager', 'rmt', 'which', and 'apt'.

Layer	Size	Command	Current Layer Contents	Filetree
75 MB	FROM d7e112e50e5fc		drwxr-xr-x	app
9.2 MB	RUN /bin/sh -c set -eux;	apt-get update; apt-get install -y --no-install-recomm	-rw-rw-r--	requirements.txt
31 MB	RUN /bin/sh -c set -eux;	savedAptMark="\$(apt-mark showmanual)"; apt-get upda	-rw-rw-rw-r	bin → /usr/bin
0 B	RUN /bin/sh -c set -eux;	for src in \$(ls pydoc3 python3 python3-config; do	drwxr-xr-x	boot
11 MB	RUN /bin/sh -c set -eux;	savedAptMark="\$(apt-mark showmanual)"; apt-get upda	drwxr-xr-x	dev
0 B	WORKDIR /app		-rw-rw-rw-r	etc
0 B	COPY requirements.txt ./ # buildkit		-rw-rw-rw-r	pwd.lock
11 MB	RUN /bin/sh -c pip install -r requirements.txt # buildkit		-rw-rw-rw-r	alternatives
252 B	COPY main.py ./ # buildkit		-rw-rw-rw-r	README

```
docker save python_good_order -o image.tar  
tar -xf image.tar  
cd 0daf5515184c3828f395b457aae06dde9da7b5e155a69c4dac09b6dce5521972  
tar -xf layer.tar  
cat ./app/requirements.txt
```

# Hadolint

<https://github.com/hadolint/hadolint>

```
docker run --rm -i ghcr.io/hadolint/hadolint < Dockerfile
```

```
-:4 DL3042 warning: Avoid use of cache directory with pip. Use `pip install --no-cache-dir <package>`
```

# ENTRYPOINT vs CMD

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
No CMD	<i>error, not allowed</i>	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry exec_cmd p1_cmd
CMD ["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

# 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.

```
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
```

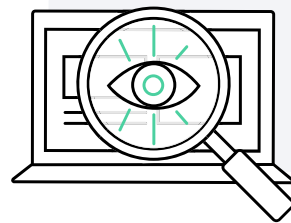
```
bash -c "$(curl -L https://setup.vector.dev)"
```

```
apt install -y vector
```

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

```
docker rm -f test_systemd_ubuntu
```

# Демонстрация работы



# Remote Docker context



3

# 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"
```

# Docker Compose Advanced



4

# Проблема мультиконтейнерных конфигурации Docker

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

Запуск вручную мультиконтейнерных 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:
```

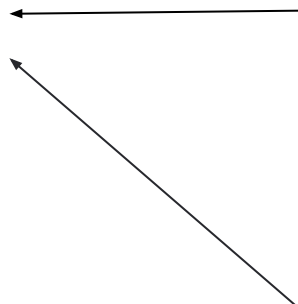
```
  image: nginx
```

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
```

Two arrows originate from the 'include' list in the main file and point to the two sub-files on the right. One arrow points from the first include entry to the top file, and the other points from the second include entry to the bottom file.

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

# 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.

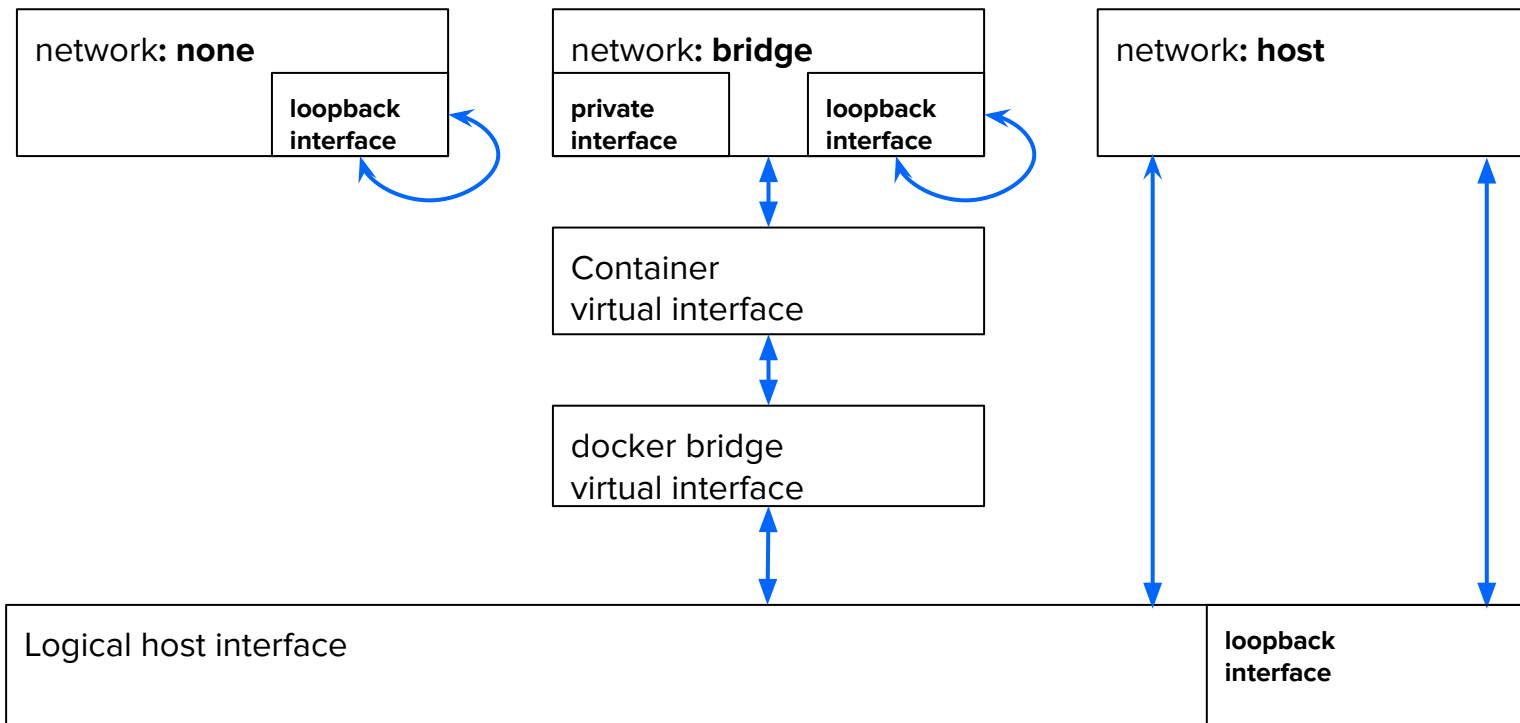
095515b8e08e	vlan	<b>ipvlan</b>
--------------	------	---------------

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

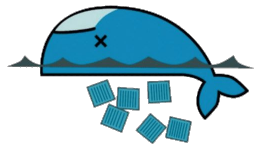


# Реалии использования Docker



5

Docker	Альтернативы
сборка Dockerfile	DOCKER_BUILDKIT; Kaniko; Buildah
docker run	podman; LXC
Develop Docker-compose	k3s; minikube; MicroK8s; Minishift
Оркестрация Docker swarm	k8s/Openshift

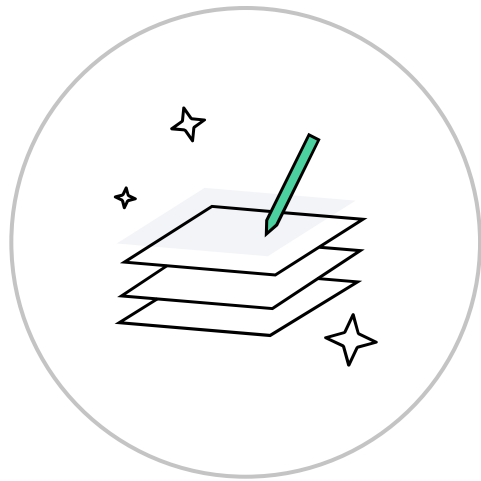




# Домашнее задание

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




# 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

<input type="checkbox"/> Дата создания	Статус	Теги	Размер	Уязвимости	Статус сканирования	Дата последнего сканирования	
<input type="checkbox"/> 04.12.2023, в 14:33	Active	3.1.3-upg1 	399.35 МБ	<div>12</div> <div>227</div> <div>357</div> <div>754</div>	Готово	04.12.2023, в 14:34	...
<input type="checkbox"/> 04.12.2023, в 14:25	Active	3.1.3 	338.65 МБ	<div>21</div> <div>334</div> <div>495</div> <div>778</div>	Готово	04.12.2023, в 14:34	...

## Pull-команда

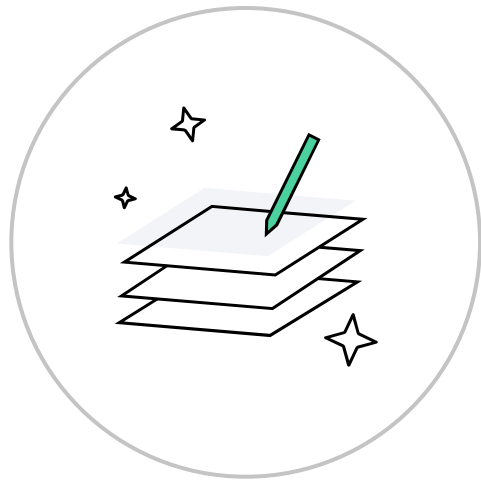
С помощью этой команды можно скачать Docker-образ из репозитория по тегу:

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

# 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/>

