

# Практическое применение docker

## часть 1

Евгений Мисяков  
SRE-инженер в Нетология



# Евгений Мисяков

О спикере:

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



# Docker “под капотом”



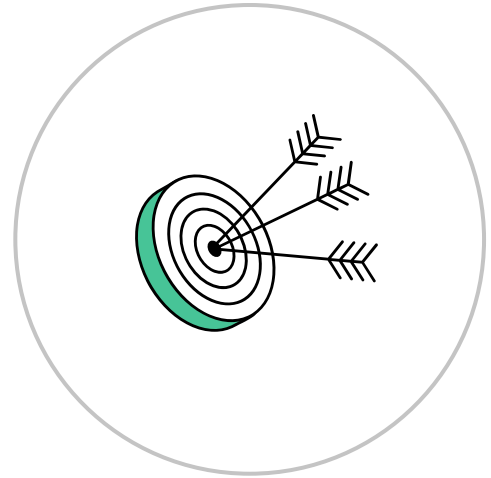
1

# 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



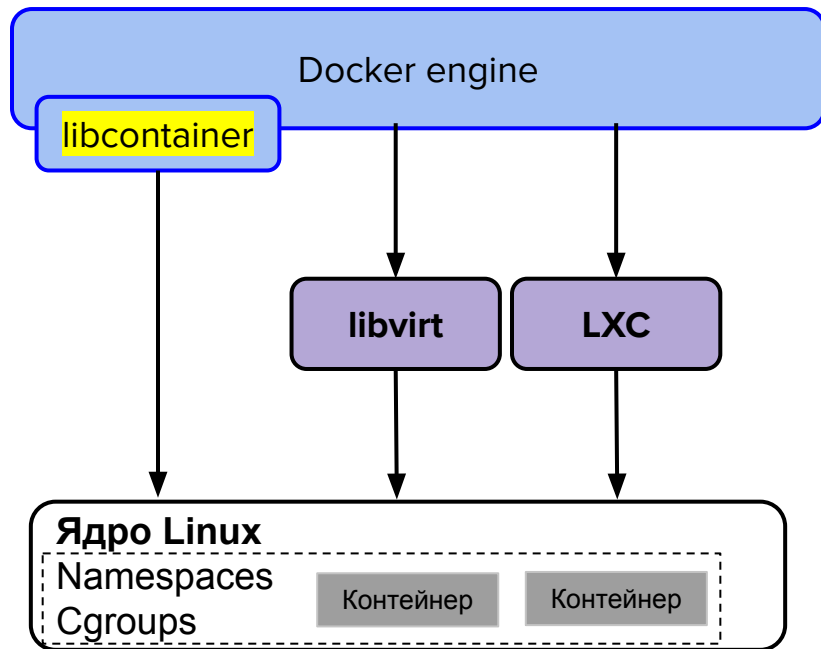
2

# 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

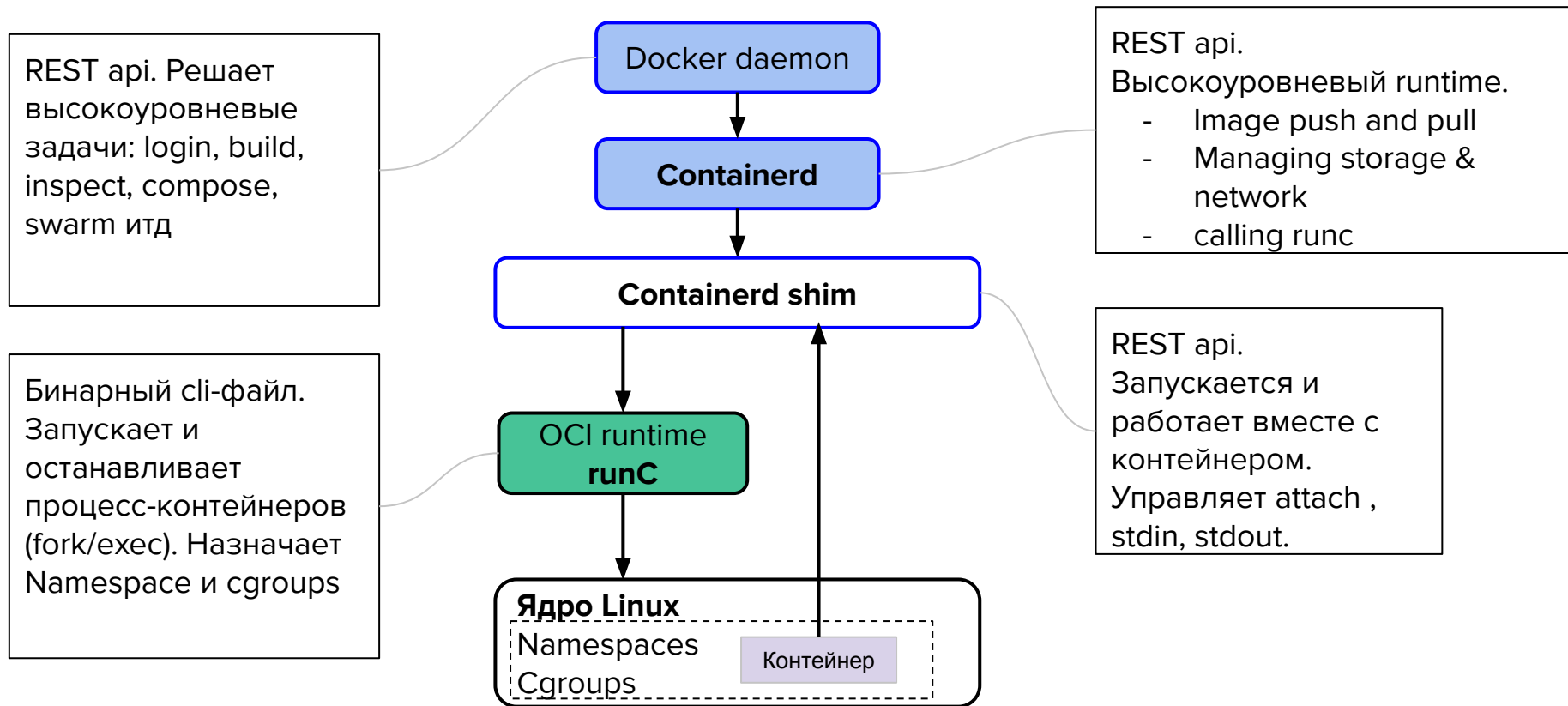
история появления контейнеризации - [ссылка](#).

# 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)



# Advanced dockerfile build and security



3

# 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!).

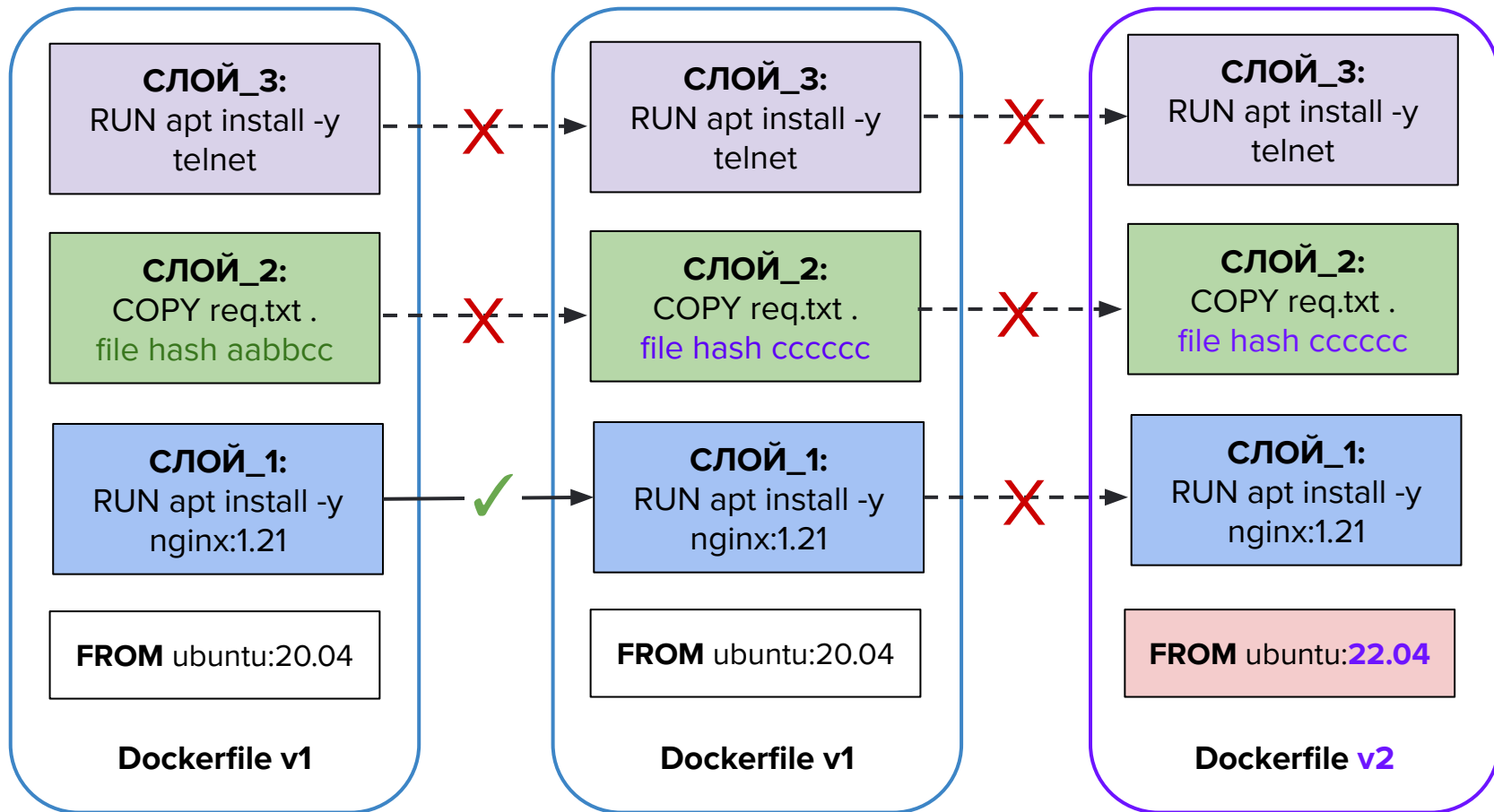
**ADD, COPY, RUN** – записывают все изменения в **новый слой**.

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

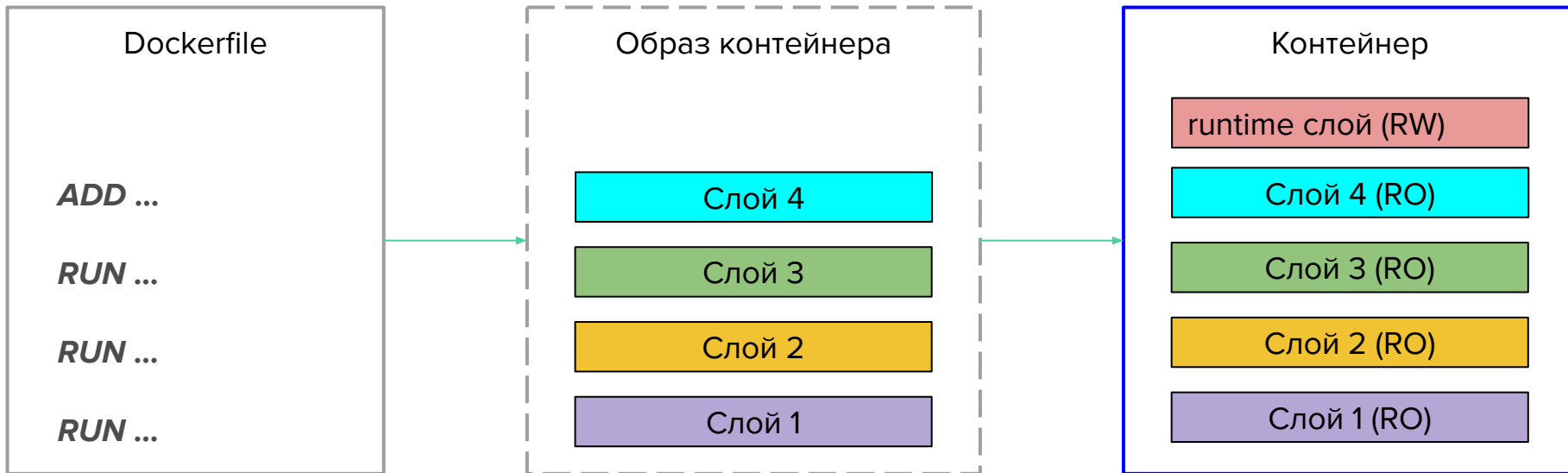
При каждой новой сборке docker builder проверяет возможность переиспользования ранее сохраненных слоев (отключается ключом --no-cache).

Правильный порядок инструкций влияет на эффективность использования кеша

# Dockerfile. Кеширование слоев



# Single-stage build или одноэтапная сборка.







## Вопрос

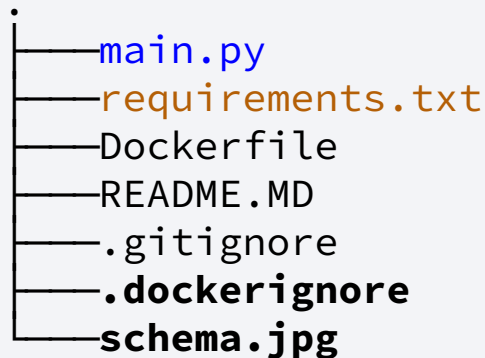
Здесь представлен правильный порядок инструкций?

```
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, with horizontal lines branching off to the right to represent files and folders. The items listed are: main.py (blue text), requirements.txt (orange text), Dockerfile (black text), README.MD (black text), .gitignore (black text), .dockerignore (bold black text), and schema.jpg (black text).

- main.py
- requirements.txt
- Dockerfile
- README.MD
- .gitignore
- .dockerignore**
- schema.jpg

**.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
```

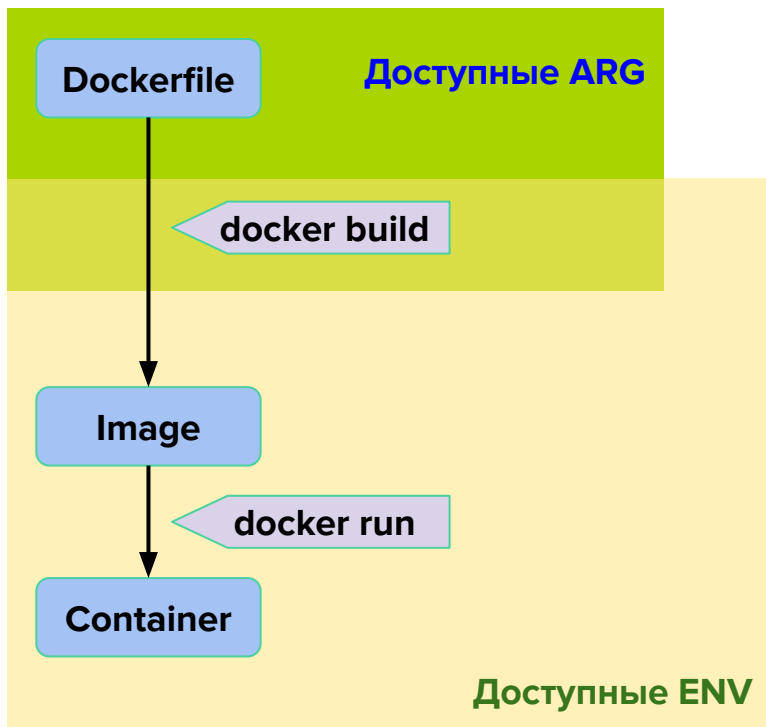
IMAGE	CREATED	CREATED BY	SIZE	COMMENT
e3270b6af7d9	6 minutes ago	CMD ["python" "main.py"]	0B	buildkit.dockerfile.v0
<missing>	6 minutes ago	COPY main.py ./ # buildkit	252B	buildkit.dockerfile.v0
<missing>	7 minutes ago	RUN /bin/sh -c pip install -r requirements.t...	11.3MB	buildkit.dockerfile.v0
<missing>	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** PASSWORD1=p@ssw0rd

**# ENV=ARG**

**ARG** PASSWORD

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

**#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 .
```

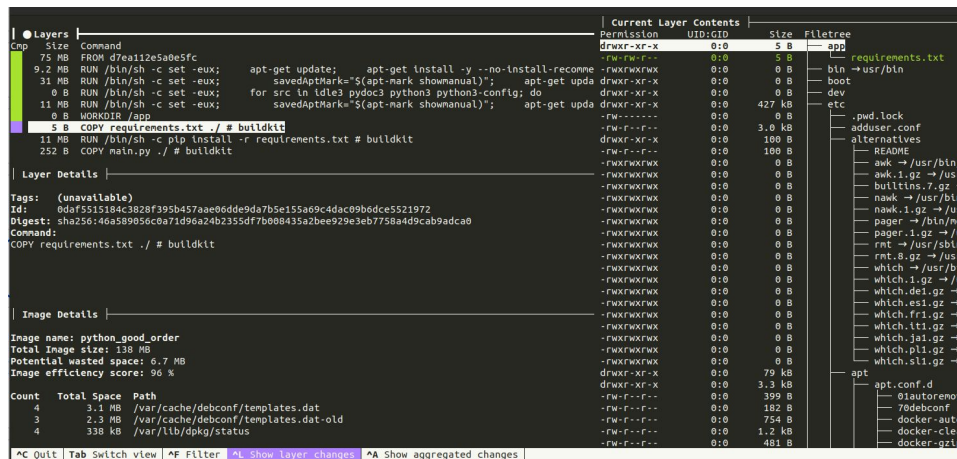
## Dockerfile:

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

# Dive, docker save

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

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



The screenshot displays the Docker Dive application interface. The left pane shows the 'Layers' section with a list of image layers, their sizes, and commands. The middle pane shows 'Layer Details' for the selected layer, including tags, IDs, digests, and the command used to create the layer. The right pane shows the 'Filetree' of the image, listing files and their sizes. The bottom pane shows 'Image Details' including the image name, total size, potential wasted space, and efficiency score. The bottom status bar shows the current layer and the ability to show aggregated changes.

Layer	Size	Command
FROM d7e112e50b5fc	75 MB	
RUN /bin/sh -c set -eux;	9.2 MB	apt-get update; apt-get install -y --no-install-recommends
RUN /bin/sh -c set -eux;	31 MB	savedAptMark="\$(apt-mark showmanual)"; apt-get update
RUN /bin/sh -c set -eux;	0 B	for src in \$(ls \$(python3 -c 'import sys; print(sys.argv[1])' requirements.txt)); do
RUN /bin/sh -c set -eux;	11 MB	savedAptMark="\$(apt-mark showmanual)"; apt-get update
COPY requirements.txt . / # buildkit	0 B	
RUN /bin/sh -c pip install -r requirements.txt # buildkit	11 MB	
COPY main.py . / # buildkit	252 B	

Layer Details

Tags: (unavailable)

Id: 0daf5515184c3828f395b457aae06dde9da7b5e155a69c4dac09b6dce5521972

Digest: sha256:46a589056ca71d96a24b2355df7b008435a2bee929e3eb7758a4d9cab9adca0

Command: COPY requirements.txt . / # buildkit

Image Details

Image name: python\_good\_order

Total Image size: 138 MB

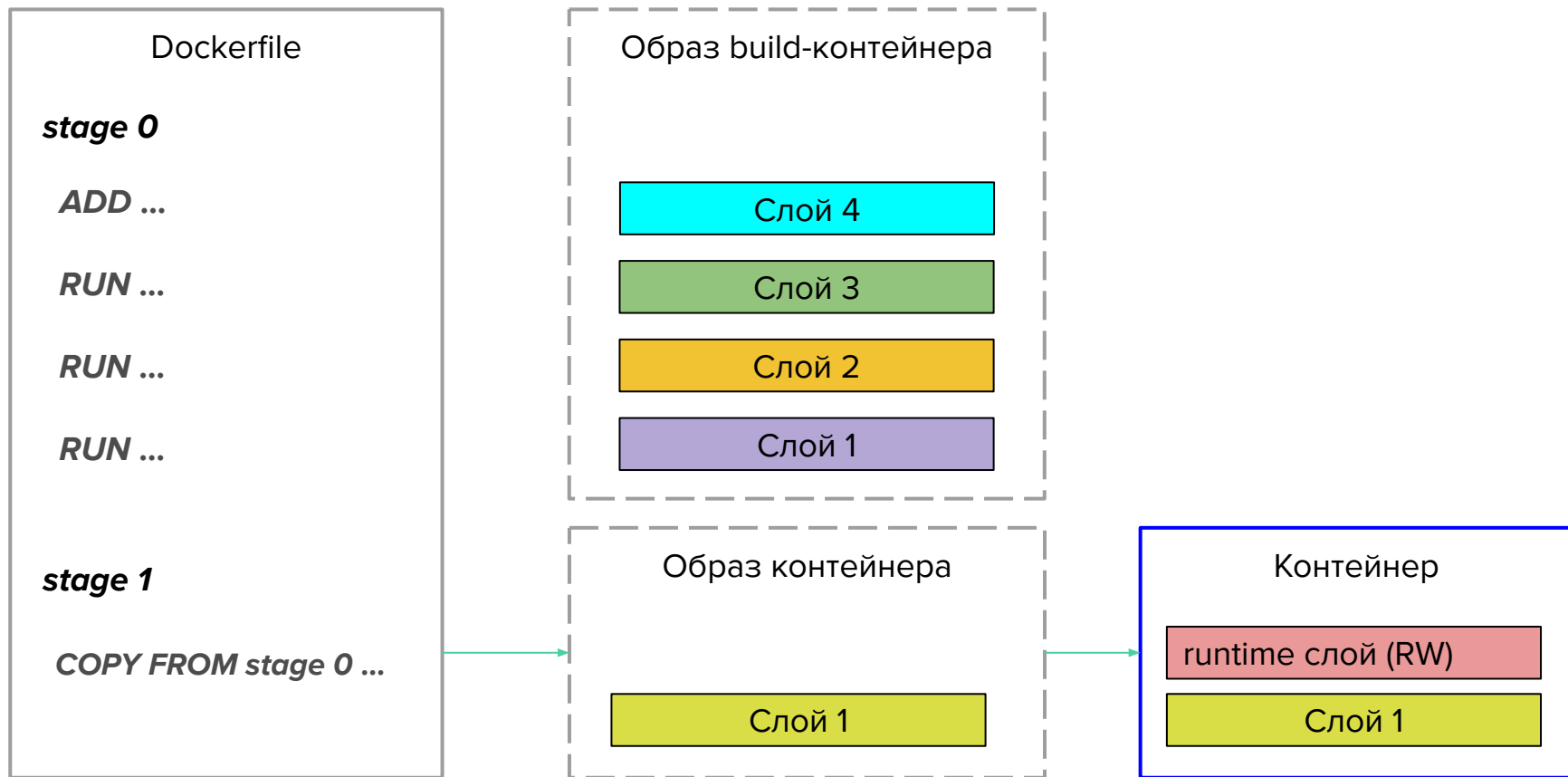
Potential wasted space: 6.7 MB

Image efficiency score: 96 %

Count	Total Space	Path
4	3.1 MB	/var/cache/debconf/templates.dat
3	2.3 MB	/var/cache/debconf/templates.dat-old
4	338 kB	/var/lib/dpkg/status

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

# Multi-stage build или многоэтапная сборка



# 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

# Демонстрация работы

