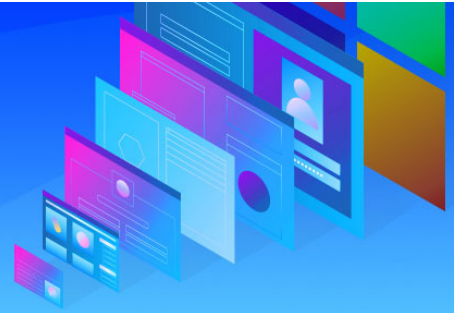


[КАК СТАТЬ АВТОРОМ](#)[Поднял стриминговую обработку – опиши в новом сезоне Java](#)

Лучшее предложение на рынке  
VPS хостинг с Windows от **523** ₽/месяц

Лицензия на ОС включена в стоимость

**1923.28**

Рейтинг

**RUVDS.com**VDS/VPS-хостинг. Скидка 15% по коду **HABR15****ru\_vds**

20 фев 2019 в 16:10

## Изучаем Docker, часть 4: уменьшение размеров образов и ускорение их сборки

8 мин 110K

Блог компании RUVDS.com, Разработка веб-сайтов\*, Виртуализация\*

[Тutorial](#)[Перевод](#)

Автор оригинала: Jeff Hale

В этой части перевода серии материалов, которая посвящена Docker, мы поговорим о том, как оптимизировать размеры образов и ускорить их сборку. В прошлых материалах мы сравнивали образы Docker с пиццей, термины с пончиками, а инструкции файлов Dockerfile с бубликами. Сегодня же не будет никакой выпечки. Пришло время посидеть на диете.

- [Часть 1: основы](#)
- [Часть 2: термины и концепции](#)
- [Часть 3: файлы Dockerfile](#)
- [Часть 4: уменьшение размеров образов и ускорение их сборки](#)
- [Часть 5: команды](#)
- [Часть 6: работа с данными](#)



Для того чтобы разобраться с тем, о чём мы будем тут говорить, вам будет полезно освежить в памяти то, о чём шла речь в третьей части этой серии материалов. А именно, там мы говорили об инструкциях файлов `Dockerfile`. Знание этих инструкций и тех особенностей Docker, которые мы обсудим сегодня, поможет вам оптимизировать файлы образов Docker.

## Кэширование

Одной из сильных сторон Docker является кэширование. Благодаря этому механизму ускоряется сборка образов.

При сборке образа Docker проходится по инструкциям файла `Dockerfile`, выполняя их по порядку. В процессе анализа инструкций Docker проверяет собственный кэш на наличие в нём образов, представляющих собой то, что получается на промежуточных этапах сборки других образов. Если подобные образы удаётся найти, то система может ими воспользоваться, не тратя время на их повторное создание.

Если кэш признан недействительным, то инструкция, в ходе выполнения которой это произошло, выполняется, создавая новый слой без использования кэша. То же самое происходит и при выполнении инструкций, которые следуют за ней.

В результате, если в ходе выполнения инструкций из `Dockerfile` оказывается, что базовый образ имеется в кэше, то используется именно этот образ из кэша. Это

называется «попаданием кэша». Если же базового образа в кэше нет, то весь процесс сборки образа будет происходить без использования кэша.

Затем следующая инструкция сопоставляется со всеми образами из кэша, в основе которых лежит тот же самый базовый образ, который уже обнаружен в кэше. Каждый кэшированный промежуточный образ проверяется на предмет того, имеется ли в нём то, что было создано такой же инструкцией. Если совпадения найти не удаётся, это называется «промахом кэша» и кэш считается недействительным. То же самое происходит до тех пор, пока не будет обработан весь файл `Dockerfile`.

Большинство новых инструкций просто сравниваются с тем, что уже есть в промежуточных образах. Если системе удаётся найти совпадение, то при сборке используется то, что уже есть в кэше.

Использование кэша способно ускорить сборку образов, но тут есть одна проблема. Например, если в `Dockerfile` обнаруживается инструкция `RUN pip install -r requirements.txt`, то Docker выполняет поиск такой же инструкции в своём локальном кэше промежуточных образов. При этом содержимое старой и новой версий файла `requirements.txt` не сравнивается.

Подобное может приводить к проблемам в том случае, если в `requirements.txt` были добавлены сведения о новых пакетах, после чего, при сборке обновлённого образа, для того, чтобы установить новый набор пакетов, нужно снова выполнить инструкцию `RUN pip install`. Совсем скоро мы поговорим о том, как бороться с этой проблемой.

В отличие от других инструкций Docker, при выполнении инструкций `ADD` и `COPY` от Docker требуется проверка содержимого файла или файлов для определения того, можно ли, при формировании образа, воспользоваться кэшем. А именно, контрольная сумма файлов, упомянутых в этих инструкциях, сравнивается с контрольной суммой файлов, которые имеются в промежуточных образах, которые уже есть в кэше. Если изменилось содержимое файлов или их метаданные, тогда кэш признаётся недействительным.

Вот несколько советов, касающихся эффективного использования кэша Docker:

- Кэширование можно отключить, передав ключ `--no-cache=True` команде `docker build`.
- Если вы собираетесь вносить изменения в инструкции `Dockerfile`, тогда каждый слой, созданный инструкциями, идущими после изменённых, будет достаточно часто собираться повторно, без использования кэша. Для того чтобы воспользоваться

преимуществами кэширования, помещайте инструкции, вероятность изменения которых высока, как можно ближе к концу Dockerfile.

- Объединяйте команды `RUN apt-get update` и `apt-get install` в цепочки для того, чтобы исключить проблемы, связанные с неправильным использованием кэша.
- Если вы используете менеджеры пакетов, наподобие `pip`, с файлом `requirements.txt`, тогда придерживайтесь нижеприведённой схемы работы для того, чтобы исключить использование устаревших промежуточных образов из кэша, содержащих набор пакетов, перечисленных в старой версии файла `requirements.txt`. Вот как это выглядит:

```
COPY requirements.txt /tmp/  
RUN pip install -r /tmp/requirements.txt  
COPY . /tmp/
```

Если вам известны другие способы борьбы с «проблемой `requirements.txt`» – можете рассказать о них в комментариях.

## Уменьшение размеров образов

### Тщательный подбор базового образа

Образы Docker могут быть довольно большими. Это противоречит вполне обоснованному стремлению того, кто их создаёт, к тому, чтобы сделать их как можно более компактными, что облегчит их загрузку из удалённого репозитория и благотворно скажется на объёме свободного места на компьютере, на который они загружаются. Поговорим о том, как уменьшать их размеры.



*Вместо бубликов и пончиков мы теперь будем есть зелень*

Одним из способов уменьшения размеров образов является тщательный подбор базовых образов и их последующая настройка.

Так, например, базовый образ Alpine представляет собой полноценный дистрибутив Linux-подобной ОС, содержащий минимум дополнительных пакетов. Его размер – примерно 5 мегабайт. Однако сборка собственного образа на основе Alpine потребует потратить достаточно много времени на то, чтобы оснастить его всем необходимым для обеспечения работы некоего приложения.

Существуют и специализированные варианты базового образа Alpine. Например, соответствующий образ из репозитория python, в который упакован скрипт `print("hello world")` весит около 78.5 Мб. Вот Dockerfile для сборки такого образа:

```
FROM python:3.7.2-alpine3.8
COPY . /app
ENTRYPOINT ["python", "./app/my_script.py", "my_var"]
```

При этом на Docker Hub сказано, что этот базовый образ имеет размер 29 Мб. Размер образа, основанного на этом базовом образе, увеличивается за счёт загрузки и



установки Python.

Помимо использования базовых образов, основанных на Alpine, уменьшить размеры образов можно благодаря использованию технологии многоступенчатой сборки.

## Многоступенчатая сборка образов

В Dockerfile, описывающем многоступенчатую сборку образа, используется несколько инструкций `FROM`. Создатель такого образа может настроить выборочное копирование файлов, называемых артефактами сборки, из одной ступени сборки в другую ступень. При этом появляется возможность избавиться от всего того, что в готовом образе не понадобится. Благодаря этому методу можно уменьшить размер готового образа.

Вот как работает каждая инструкция `FROM`:

- Она начинает новый шаг сборки.
- Она не зависит от того, что было создано на предыдущем шаге сборки.
- Она может использовать базовый образ, отличающийся от того, который применялся на предыдущем шаге.

Вот модифицированный пример файла Dockerfile из документации Docker, описывающего многоступенчатую сборку.

```
FROM golang:1.7.3 AS build
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .
FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=build /go/src/github.com/alexellis/href-counter/app .
CMD ["/app"]
```

Обратите внимание на то, что мы дали имя первой ступени сборки, указав его после инструкции `FROM`. К именованному этапу сборки мы обращаемся в инструкции `COPY --from=` ниже в Dockerfile.

Применение процесса многоступенчатой сборки образов имеет смысл в некоторых случаях,

когда приходится создавать множество контейнеров для продакшн-окружения.

Многоступенчатая сборка позволяет максимально сократить размеры готовых образов. Но иногда такой подход приводит к усложнению поддержки образов. Поэтому вы, вероятно, не будете пользоваться многоступенчатой сборкой образов в тех случаях, в которых без неё можно обойтись. Об особенностях этой технологии можно почитать [здесь](#) и [здесь](#).

Как видите, многоступенчатая сборка – технология интересная, но подходит она далеко не для всех случаев. Тот же способ уменьшения размера образов, который мы обсудим ниже, можно порекомендовать абсолютно всем.

## | Файл `.dockerignore`

О файлах `.dockerignore` нужно знать абсолютно всем, кто хочет освоить Docker. Эти файлы похожи на файлы `.gitignore`. Они содержат список файлов и папок, в виде имён или шаблонов, которые Docker должен игнорировать в ходе сборки образа.

Этот файл размещают там же, где находится файл `Dockerfile`, и всё остальное, входящее в контекст сборки образа.

При запуске команды `docker build`, иницилирующей сборку образа, Docker проверяет папку на наличие в ней файла `.dockerignore`. Если такой файл найти удаётся, тогда этот файл разбирается, при этом при определении списка файлов, которые нужно игнорировать, используются правила функции `Match()` из пакета `filepath` Go и некоторые собственные правила Docker.

Так, например, если в файле `.dockerignore` встретится шаблон вида `*.jpg`, то при создании образа проигнорированы будут файлы с любым именем и с расширением `.jpg`. Если в файле встретится строка `videos`, то система проигнорирует папку `videos` и всё её содержимое.

При составлении файла `.dockerignore` его можно снабжать комментариями, используя символ `#`.

Вот что даёт тому, кто занимается созданием образов Docker, применение файлов `.dockerignore`:

- Это позволяет исключать из состава образа файлы, содержащие секретные сведения наподобие логинов и паролей.
- Это позволяет уменьшить размер образа. Чем меньше в образе файлов – тем меньше будет его размер и тем быстрее с ним можно будет работать.

- Это даёт возможность уменьшить число поводов для признания недействительным кэша при сборке похожих образов. Например, если при повторной сборке образа меняются некие служебные файлы проекта, наподобие файлов с журналами, из-за чего данные, хранящиеся в кэше, по сути, необоснованно признаются недействительными, это замедляет сборку образов.

Подробности о файле `.dockerignore` можно почитать в документации к Docker.

## Исследование размеров образов

Поговорим о том, как, пользуясь средствами командной строки, узнавать размеры образов и контейнеров Docker.

- Для того чтобы выяснить примерный размер выполняющегося контейнера, можно использовать команду вида `docker container ls -s`.
- Команда `docker image ls` выводит размеры образов.
- Узнать размеры промежуточных образов, из которых собран некий образ, можно с помощью команды `docker image history my_image:my_tag`.
- Команда `docker image inspect my_image:tag` позволяет узнать подробные сведения об образе, в том числе – размер каждого его слоя. Слои немного отличаются от промежуточных образов, из которых состоит готовый образ, но, в большинстве случаев их можно рассматривать как одинаковые сущности. Вот хороший материал, который посвящён подробностям внутреннего устройства образов Docker.
- Для того чтобы исследовать содержимое контейнеров можно установить пакет `dive`.

Теперь, когда мы обсудили возможности по уменьшению размеров образов, предлагаю вашему вниманию восемь рекомендаций, касающихся уменьшения размеров образов и ускорения процесса их сборки.

## Рекомендации по уменьшению размеров образов и ускорению процесса их сборки

1. Используйте всегда, когда это возможно, официальные образы в качестве базовых образов. Официальные образы регулярно обновляются, они безопаснее неофициальных образов.
2. Для того чтобы собирать как можно более компактные образы, пользуйтесь базовыми образами, основанными на Alpine Linux.



3. Если вы пользуетесь `apt`, комбинируйте в одной инструкции `RUN` команды `apt-get update` и `apt-get install`. Кроме того, объединяйте в одну инструкцию команды установки пакетов. Перечисляйте пакеты в алфавитном порядке на нескольких строках, разделяя список символами `\`. Например, это может выглядеть так:

```
RUN apt-get update && apt-get install -y \  
    package-one \  
    package-two \  
    package-three  
&& rm -rf /var/lib/apt/lists/*
```

Этот метод позволяет сократить число слоёв, которые должны быть добавлены в образ, и помогает поддерживать код файла в приличном виде.

4. Включайте конструкцию вида `&& rm -rf /var/lib/apt/lists/*` в конец инструкции `RUN`, используемой для установки пакетов. Это позволит очистить кэш `apt` и приведёт к тому, что он не будет сохраняться в слое, сформированном командой `RUN`. Подробности об этом можно почитать в документации.
5. Разумно пользуйтесь возможностями кэширования, размещая в `Dockerfile` команды, вероятность изменения которых высока, ближе к концу файла.
6. Пользуйтесь файлом `.dockerignore`.
7. Взгляните на `dive` – отличный инструмент для исследования образов Docker, который помогает в деле уменьшения их размеров.
8. Не устанавливайте в образы пакеты, без которых можно обойтись.

## Итоги

Теперь вы знаете о том, как сделать так, чтобы образы Docker быстро собирались бы, быстро загружались бы из репозитория и не занимали бы слишком много места на компьютере. В следующий раз мы поговорим о командах Docker.

**Уважаемые читатели!** Сталкивались ли вы при сборке образов Docker с проблемами, связанными с неправильным использованием механизмов кэширования?

**Habrahabr10**

Промо-код для скидки в 10% на наши виртуальные сервера

**Теги:** Docker, разработка


**Хабы:** Блог компании RUVDS.com, Разработка веб-сайтов, Виртуализация

## Редакторский дайджест



Присылаем лучшие статьи раз в месяц


Электронпочта



RUVDS.com

VDS/VPS-хостинг. Скидка 15% по коду **HABR15**

Telegram ВКонтакте Twitter



409

349.3

Карма

Рейтинг

@ru\_vds  
Пользователь


 Комментарии 1


## Публикации


ЛУЧШИЕ ЗА СУТКИ    ПОХОЖИЕ


22 часа назад


### Герои напильника и паяльника: итоги сезона DIY


 8 мин


 8.5K


 Сезон DIY

 Спецпроект

 +44


 42


 6


 spiritus\_sancti


23 часа назад

### RGB-усилители. Особенности, проблемы, выбор

 Простой

 6 мин

 5.6K

 Тutorial

 +40 24 11

ru\_vds

19 часов назад

## Профилирование Python — почему и где тормозит ваш код



Средний



10 мин



3.6K

Тutorial

Перевод

 +32 93 4

BiktorSergeev

16 часов назад

## WebOne: даём жизнь старым браузерам



6 мин



3.3K

 +30 29 11

zatim

1 час назад

## Видеокарта VGA для микроконтроллера



Средний



17 мин



911

Тutorial

 +21 18 7

it\_union

2 часа назад

## ЗАО Гейм Инсайт Труп



Простой



4 мин



3.4K

 +21 7 7

AnnaVIMorozova

19 часов назад

## Как повысить эффективность коммуникаций в команде: учимся решать конфликты



7 мин



2.3K

+19

43

2



Yu-Leo  
вчера в 15:22

Обзор электронной книги Meebook P10 Pro

Простой 9 мин 6.4K

Обзор

+18

18

8



Sagidullin  
6 часов назад

Всего два месяца — и новый релиз ядра Linux. Что появилось в ядре 6.5, что изменилось и что удалили. Новые возможности

5 мин 5.3K

+15

6

5



mr-pickles  
22 часа назад

Архетипы программных архитекторов. Часть 2

Простой 9 мин 2.5K

Перевод

+15

32

0

Показать еще

ИНФОРМАЦИЯ

Ваш аккаунт

Войти  
Регистрация

Разделы

Статьи  
Новости  
Хабы  
Компании  
Авторы

Информация

Устройство сайта  
Для авторов  
Для компаний  
Документы  
Соглашение

Услуги

Корпоративный блог  
Медийная реклама  
Нативные проекты  
Образовательные программы



- Настройка языка
- Техническая поддержка
- Вернуться на старую версию

© 2006–2023, Habr

.....

VPS Windows от 523 рублей в месяц. Бесплатный тестовый период 3 дня.  
ruvds.com


VDS в Цюрихе. Дата-центр TIER III – швейцарское качество по низкой цене.  
ruvds.com

Антивирусная защита виртуального сервера. Легкий агент для VPS.  
ruvds.com

VPS в Лондоне. Дата-центр TIER III – английская точность за рубли.  
ruvds.com

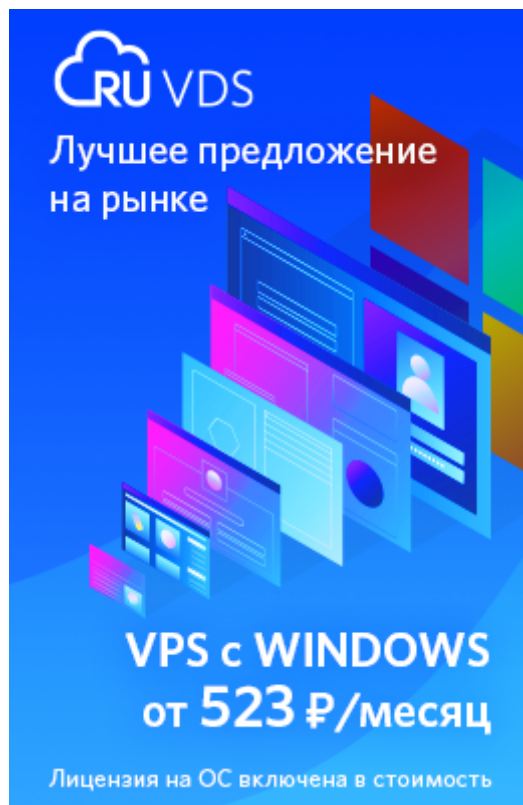
VPS с видеокартой на мощных серверах 3,4ГГц  
ruvds.com

ПРИЛОЖЕНИЯ

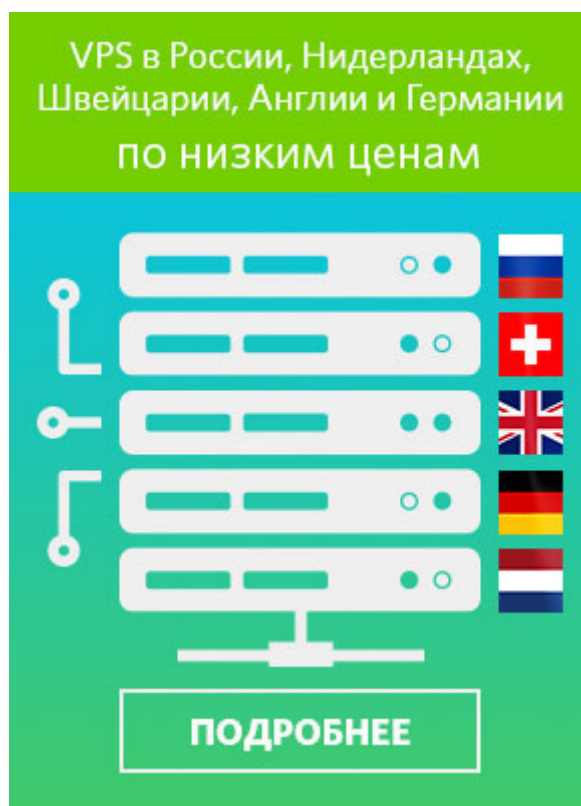


**RUVDS Client**  
Приложение для мониторинга и управления виртуальными серверами RUVDS с мобильных устройств.  
Android iOS

ВИДЖЕТ



## ВИДЖЕТ



## БЛОГ НА ХАБРЕ

19 часов назад

<https://habr.com/ru/companies/ruvds/articles/440658/>



## Профилирование Python — почему и где тормозит ваш код

 3.6K  4

---

23 часа назад

## RGB-усилители. Особенности, проблемы, выбор

 5.6K  11

---

27 авг в 17:00

## Интернет 90-х: когда после 20 часов в онлайн тебе пишет президент ISP

 15K  40

---

26 авг в 17:00

## История компьютерных стратегий. Часть 8. «Age of Empires»: шедевр геймдева, от которого бомбит у любителей истории

 13K  12

---

25 авг в 16:00

## Xbox is a new Dreamcast. Зачем покупать консоль от Microsoft в 2023 году и во что играть

 6.5K  8