



Docker:

Нюансы при разработке на разных
языках программирования

Спикер: Александр Швалов

Языки программирования и Docker

- Для тех языков программирования, которые уже давно существуют, с Docker может быть не просто разрабатывать.

Языки программирования и Docker

- Для тех языков программирования, которые уже давно существуют, с Docker может быть не просто разрабатывать.
- Популярность контейнеров за несколько лет привела к выработке лучших решений для написания программ и ведения разработки.

Языки программирования и Docker

- Для тех языков программирования, которые уже давно существуют, с Docker может быть не просто разрабатывать.
- Популярность контейнеров за несколько лет привела к выработке лучших решений для написания программ и ведения разработки.
- Есть общие потребности, актуальные для всех:

Языки программирования и Docker

- Для тех языков программирования, которые уже давно существуют, с Docker может быть непросто разрабатывать.
- Популярность контейнеров за несколько лет привела к выработке лучших решений для написания программ и ведения разработки.
- Есть общие потребности, актуальные для всех:
 1. упрощение отладки при разработке;

Языки программирования и Docker

- Для тех языков программирования, которые уже давно существуют, с Docker может быть непросто разрабатывать.
- Популярность контейнеров за несколько лет привела к выработке лучших решений для написания программ и ведения разработки.
- Есть общие потребности, актуальные для всех:
 1. упрощение отладки при разработке;
 2. сокращение времени на сборку образа;

Языки программирования и Docker

- Для тех языков программирования, которые уже давно существуют, с Docker может быть непросто разрабатывать.
- Популярность контейнеров за несколько лет привела к выработке лучших решений для написания программ и ведения разработки.
- Есть общие потребности, актуальные для всех:
 1. упрощение отладки при разработке;
 2. сокращение времени на сборку образа;
 3. уменьшение размера получившегося образа.

Как улучшить процесс разработки?

- Монтирование каталога с кодом в контейнер при отладке.



Как улучшить процесс разработки?

- Монтирование каталога с кодом в контейнер при отладке.
- Копирование в образ самых часто изменяемых элементов в последнюю очередь.



Как улучшить процесс разработки?

- Монтирование каталога с кодом в контейнер при отладке.
- Копирование в образ самых часто изменяемых элементов в последнюю очередь.
- Использование многоступенчатой сборки.



Python

- Скриптовый язык. Для выполнения нужен интерпретатор.



Python

- Скриптовый язык. Для выполнения нужен интерпретатор.
- Docker упрощает работу с версиями языка и pip-пакетов, поэтому важно указывать конкретные версии образов в директиве FROM, а также точные версии пакетов в requirements.txt



Python

- Скриптовый язык. Для выполнения нужен интерпретатор.
- Docker упрощает работу с версиями языка и pip-пакетов, поэтому важно указывать конкретные версии образов в директиве FROM, а также точные версии пакетов в requirements.txt
- Первым делом копируем в образ requirements.txt и ставим зависимости, а уже потом копируем код (или монтируем каталог с хоста).



Python

- Скриптовый язык. Для выполнения нужен интерпретатор.
- Docker упрощает работу с версиями языка и pip-пакетов, поэтому важно указывать конкретные версии образов в директиве FROM, а также точные версии пакетов в requirements.txt
- Первым делом копируем в образ requirements.txt и ставим зависимости, а уже потом копируем код (или монтируем каталог с хоста).
- Лучше разделять миграции БД и запуск приложения. Для этого не требуется собирать отдельный образ, можно управлять поведением с помощью переменных окружения при каждом запуске.



Ещё немного о Python

- Если вам всё же нужно активировать виртуальное окружение **virtualenv**, элегантным решением будет задать переменную окружения в **Dockerfile**.



Ещё немного о Python

- Если вам всё же нужно активировать виртуальное окружение **virtualenv**, элегантным решением будет задать переменную окружения в **Dockerfile**.
- При использовании **gunicorn** в качестве веб-сервера в Docker, можете задать такие параметры запуска:
 - `gunicorn --worker-tmp-dir /dev/shm` — нужно для ускорения запуска и работы, так как по умолчанию используется каталог /tmp;



Ещё немного о Python

- Если вам всё же нужно активировать виртуальное окружение **virtualenv**, элегантным решением будет задать переменную окружения в **Dockerfile**.
- При использовании **gunicorn** в качестве веб-сервера в Docker, можете задать такие параметры запуска:
 - `gunicorn --worker-tmp-dir /dev/shm` — нужно для ускорения запуска и работы, так как по умолчанию используется каталог /tmp;
 - `gunicorn --workers=2 --threads=4 --worker-class=gthread` — если воркер будет всего один, есть риск перезапуска контейнера при медленном ответе на запрос;



Ещё немного о Python

- Если вам всё же нужно активировать виртуальное окружение **virtualenv**, элегантным решением будет задать переменную окружения в **Dockerfile**.
- При использовании **gunicorn** в качестве веб-сервера в Docker, можете задать такие параметры запуска:
 - **gunicorn --worker-tmp-dir /dev/shm** — нужно для ускорения запуска и работы, так как по умолчанию используется каталог /tmp;
 - **gunicorn --workers=2 --threads=4 --worker-class=gthread** — если воркер будет всего один, есть риск перезапуска контейнера при медленном ответе на запрос;
 - **gunicorn --log-file=-** - перенаправление логов в STDOUT/STDERR.



Python Dockerfile, два примера:

```
# Первый шаг
FROM python:3.8 AS builder
COPY requirements.txt .

# Устанавливаем зависимости в локальный каталог
# (например /root/.local)
RUN pip install --user -r requirements.txt

# Второй шаг, уже без имени
FROM python:3.8-slim
WORKDIR /code

# Копируем только зависимости из образа,
# полученного на первом шаге
COPY --from=builder /root/.local/bin /root/.local
COPY ./src .

# Устанавливаем переменную окружения PATH
ENV PATH=/root/.local:$PATH

CMD [ "python", "./server.py" ]
```

```
FROM python:3.8-slim-buster
```

```
# Устанавливаем переменные окружения для
«активации» виртуального окружения
ENV VIRTUAL_ENV=/opt/venv
RUN python3 -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"
```

```
# Устанавливаем зависимости
COPY requirements.txt .
RUN pip install -r requirements.txt
```

```
# Запускаем приложение, установленные переменные
# окружения из шагов выше всё ещё действуют
COPY myapp.py .
CMD [ "python", "myapp.py" ]
```



Java

- Если у вас legacy-приложение, есть риски: при использовании, например, 7-й версии, java-машина не понимала ограничения cgroups.



Java

- Если у вас legacy-приложение, есть риски: при использовании, например, 7-й версии, java-машина не понимала ограничения cgroups.
- Для сборки используется один тип образов (например, с Maven), для запуска – другой (openjdk).



Java

- Если у вас legacy-приложение, есть риски: при использовании, например, 7-й версии, java-машина не понимала ограничения cgroups.
- Для сборки используется один тип образов (например, с Maven), для запуска – другой (openjdk).
- Для повторного использования кэша Maven можно подключать том, где кэш будет храниться между запусками контейнера.



Пример Java Dockerfile

```
# Первый шаг сборки
FROM maven:3.6-jdk-8-alpine AS builder

ARG USER_HOME_DIR="/root"
VOLUME "$USER_HOME_DIR/.m2"
ENV MAVEN_CONFIG "$USER_HOME_DIR/.m2"
WORKDIR /app

# Копируем зависимости и устанавливаем их
COPY pom.xml .
RUN mvn -e -B dependency:resolve

# Сборка jar-файла
COPY src ./src
RUN mvn -e -B package

# Второй шаг, копирование приложения и его запуск
FROM openjdk:8-jre-alpine
COPY --from=builder /app/target/app.jar /
CMD ["java", "-jar", "/app.jar"]
```



PHP

- Для PHP нужен интерпретатор и веб-сервер, то есть запущенных в контейнере приложений может получиться больше одного.



PHP

- Для PHP нужен интерпретатор и веб-сервер, то есть запущенных в контейнере приложений может получиться больше одного.
- Nginx и apache конфигурируются с помощью файлов и пишут логи в файлы, поэтому не очень дружелюбны к Docker.



PHP

- Для PHP нужен интерпретатор и веб-сервер, то есть запущенных в контейнере приложений может получиться больше одного.
- Nginx и apache конфигурируются с помощью файлов и пишут логи в файлы, поэтому не очень дружелюбны к Docker.
- Поэтому PHP в Docker чаще всего используют для разработки, а не в продакшене. Например, для быстрого запуска окружений nginx + php-fpm + composer + mysql.



PHP Dockerfile (на примере Laravel):

```
FROM php:7.2-fpm

COPY composer.lock composer.json /var/www/
WORKDIR /var/www
RUN apt-get update && apt-get install -y \
    build-essential \
    libpng-dev \
    libjpeg62-turbo-dev \
    libfreetype6-dev \
    locales \
    zip \
    jpegoptim optipng pngquant gifsicle \
    vim \
    unzip \
    git \
    curl

RUN apt-get clean && rm -rf /var/lib/apt/lists/*

RUN docker-php-ext-install pdo_mysql mbstring zip
exif pcntl
```

```
RUN docker-php-ext-configure gd --with-gd --with-
freetype-dir=/usr/include/ --with-jpeg-
dir=/usr/include/ --with-png-dir=/usr/include/
RUN docker-php-ext-install gd
```

```
RUN curl -sS https://getcomposer.org/installer |
php -- --install-dir=/usr/local/bin --
filename=composer
```

```
RUN groupadd -g 1000 www
RUN useradd -u 1000 -ms /bin/bash -g www www
```

```
COPY . /var/www
```

```
COPY --chown=www:www . /var/www
```

```
USER www
```

```
EXPOSE 9000
CMD ["php-fpm"]
```



Ruby

- Неплохо отвечает требованиям Docker, приложения настраиваются с помощью переменных окружения.



Ruby

- Неплохо отвечает требованиям Docker, приложения настраиваются с помощью переменных окружения.
- Типичный стэк выглядит так: postgres, redis, sidekiq, nginx и само приложение – итого получится 5 контейнеров.



Ruby

- Неплохо отвечает требованиям Docker, приложения настраиваются с помощью переменных окружения.
- Типичный стек выглядит так: postgres, redis, sidekiq, nginx и само приложение – итого получится 5 контейнеров.
- Для автоматического перечитывания изменённых файлов можно запустить в одном контейнере с **rails server** ещё и **guard**. Это упростит отладку фронтенда.



Ruby

- Неплохо отвечает требованиям Docker, приложения настраиваются с помощью переменных окружения.
- Типичный стэк выглядит так: postgres, redis, sidekiq, nginx и само приложение – итого получится 5 контейнеров.
- Для автоматического перечитывания изменённых файлов можно запустить в одном контейнере с **rails server** ещё и **guard**. Это упростит отладку фронтенда.
- Для приближения окружения к продакшену можно запустить в одном контейнере rails и nginx.



Ещё немного о Ruby

- При установке зависимостей можно использовать конструкцию

```
RUN bundle check || bundle install
```



Ещё немного о Ruby

- При установке зависимостей можно использовать конструкцию

```
RUN bundle check || bundle install
```

- Для исключения проблем с pid-файлом при запуске rails можно хранить файл в примонтированном tmpfs, либо запускать bundle скриптом, который вначале проверяет наличие pid-файла и удаляет его.



Ещё немного о Ruby

- При установке зависимостей можно использовать конструкцию

```
RUN bundle check || bundle install
```

- Для исключения проблем с pid-файлом при запуске rails можно хранить файл в примонтированном tmpfs, либо запускать bundle скриптом, который вначале проверяет наличие pid-файла и удаляет его.
- Для уменьшения размера образа можно удалить статику и кэши.



Ruby on Rails Dockerfile

```
FROM ruby:2.3-alpine

RUN set -ex && apk add --update --virtual runtime-deps postgresql-client nodejs
libffi-dev readline sqlite && rm -rf /var/cache/apk/*

WORKDIR /tmp
COPY Gemfile Gemfile.lock ./

RUN set -ex && apk add --virtual build-deps build-base openssl-dev postgresql-
dev libc-dev linux-headers libxml2-dev libxslt-dev readline-dev && \
    bundle install --clean --no-cache --without development --jobs=2 && \
    rm -rf /var/cache/apk/* && \
    apk del build-deps

ENV APP_HOME /app
COPY . $APP_HOME
WORKDIR $APP_HOME

ENV RAILS_ENV=production RACK_ENV=production

EXPOSE 3000

CMD ["bundle", "exec", "puma", "-C", "config/puma.rb"]
```



Golang

- Docker сам написан на Go, поэтому этот язык является одним из лучших для использования в контейнерах.



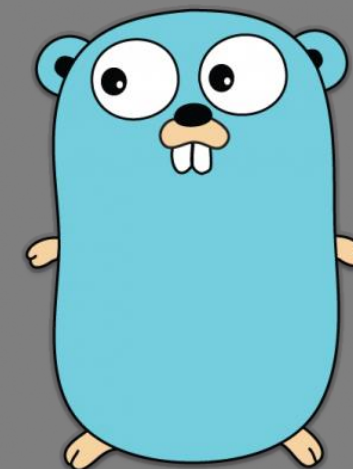
Golang

- Docker сам написан на Go, поэтому этот язык является одним из лучших для использования в контейнерах.
- Это компилируемый язык, а значит для запуска бинарного файла мы можем использовать образ **FROM scratch**.



Golang

- Docker сам написан на Go, поэтому этот язык является одним из лучших для использования в контейнерах.
- Это компилируемый язык, а значит для запуска бинарного файла мы можем использовать образ **FROM scratch**.
- Однако для упрощения команд при локальной сборке зачастую применяется старый добрый Makefile.



Golang

- Docker сам написан на Go, поэтому этот язык является одним из лучших для использования в контейнерах.
- Это компилируемый язык, а значит для запуска бинарного файла мы можем использовать образ **FROM scratch**.
- Однако для упрощения команд при локальной сборке зачастую применяется старый добрый Makefile.
- Всё так же актуальна многоступенчатая сборка образов.
- Moby - это опенсорсный проект Docker:
<https://github.com/moby/moby#the-moby-project>



Golang Dockerfile

```
# Первый шаг сборки
FROM golang:alpine AS builder

# Ставим git, это нужно для установки зависимостей
RUN apk update && apk add --no-cache git

ENV USER=appuser
ENV UID=10001

# Создаём пользователя максимально безопасно
RUN adduser \
    --disabled-password \
    --gecos "" \
    --home "/nonexistent" \
    --shell "/sbin/nologin" \
    --no-create-home \
    --uid "${UID}" \
    "${USER}" WORKDIR $GOPATH/src/mypackage/myapp/

# Копируем код в образ и ставим зависимости
COPY . .
RUN go get -d -v
```

```
# Если используете Go 1.11 и выше
# RUN go mod download
# RUN go mod verify

# Запускаем оптимизированную сборку
RUN GOOS=linux GOARCH=amd64 go build -ldflags="-w -s" -o /go/bin/hello

# Второй шаг сборки
FROM scratch

# Копируем всё необходимое из первого шага
COPY --from=builder /etc/passwd /etc/passwd
COPY --from=builder /etc/group /etc/group
COPY --from=builder /go/bin/hello /go/bin/hello

# Простой пользователь
USER appuser:appuser

# Порт с номером более 1024
EXPOSE 9292

ENTRYPOINT ["/go/bin/hello"]
```



СЛЁРМ



slurm.io



Southbridge



southbridge.io

* все использованные логотипы языков программирования
принадлежат их правообладателям