Docker

Команда FAANG School приветствует тебя и рада представить то, чего все так давно просили — подробнейшее руководство по Docker!

Мануал будет полезен как для начинающих разработчиков, так и для профессионалов индустрии и содержит объяснение основных команд и концепций Docker.

Что вас ждет:

- изучите создание и управление образами Docker;
- разберетесь с управлением контейнерами и освоите синтаксис Dockerfile;
- изучите Docker Compose, а также освоите новейшие функции, такие как Docker Compose Watch, с помощью понятных объяснений, пошаговых инструкций и практичных иллюстрированных примеров.

Благодаря всем необходимым командам и концепциям этот гайд поможет достаточно быстро начать работу с Docker!

Но хватит слов, переходим к самому интересному!

Этот мануал создан, чтобы **быстрее помочь** тебе разобраться с Docker, его основными операциями и синтаксисом!

А представь, что ты не просто изучаешь теорию, а **сразу применяешь** полученные знания на практике **в проектах**, востребованных в **крупнейших IT-компаниях!** Звучит здорово, не правда ли?!

Но где найти такое место?!
В FAANG School, конечно же, а точнее, на легендарном **Java Буткемп!**

- **За 5 месяцев** интенсивного обучения освоишь необходимые навыки современному и востребованному Java-разработчику!
- **Создашь 9 микросервисов** для своего портфолио — мощный проект, который продемонстрируют твои знания и навыки работодателям.
- **Получишь незаменимый опыт работы в команде**, максимально приближенный к условиям реальных IT-проектов
- **Создашь крутое резюме**, которое выделит тебя среди сотен других кандидатов!
- **Научишься писать чистый код** благодаря неограниченному код-ревью наших менторов и техлидов!

Хочешь уже сейчас начать карьеру в IT, но не понимаешь с чего начинать? Записывайся на БЕСПЛАТНУЮ консультацию от команды FAANG School, где мы дадим индивидуальные рекомендации по развитию твоей карьеры!

Шпаргалка

no Dockerfile

Что такое Dockerfile?

Dockerfile — это скрипт, который содержит инструкции для создания образа Docker. Он определяет базовый образ для создания вашего собственного, устанавливает переменные среды, программное обеспечение и настраивает контейнер для конкретного приложения или сервиса.

Синтаксис Dockerfile

FROM:

Указывает базовый образ для образа Docker.

```
FROM image_name:tag
#Example
FROM ubuntu:20.04
```

WORKDIR:

Устанавливает рабочую директорию (папку) для последующих инструкций.

```
WORKDIR /path/to/directory
#Example
WORKDIR /app
```

COPY:

Копирует файлы или каталоги из контекста сборки образа в контейнер. Из ОС, где лежит Dockerfile, на ОС образа, который собирает этот Dockerfile.

```
COPY host_source_path container_destination_path
#Example
COPY ..
```

RUN:

Выполняет консольные команды на ОС образа.

```
RUN command1 && command2
#Example
RUN apt-get update && apt-get install -y curl
```

ENV:

Устанавливает переменные окружения в образе.

```
ENV KEY=VALUE
#Example
ENV NODE_VERSION=14
```

EXPOSE:

Сообщает Docker, что контейнер слушает указанные сетевые порты во время выполнения.

```
EXPOSE port
#Example
EXPOSE 8080
```

CMD:

Предоставляет консольные команды или параметры для запуска по умолчанию для выполняемого контейнера.

```
CMD ["executable","param1","param2"]
#Example
CMD ["npm", "start"]
```

или

```
CMD executable param1 param2
#Example
CMD npm run dev
```

ARG:

Определяет переменные, которые пользователи могут передать на этапе сборки билдеру с помощью команды docker build.

```
ARG VARIABLE_NAME=default_value
#Example
ARG VERSION=latest
```

ENTRYPOINT:

Настроить контейнер для выполнения как исполняемого файла. Устанавливает переменные окружения в образе.

```
ENTRYPOINT ["executable", "param1", "param2"]
#Example
ENTRYPOINT ["node", "app.js"]
```

или

```
ENTRYPOINT executable param1 param2
#Example
ENTRYPOINT node app.js
```

VOLUME:

Создаёт точку размещения внешних volumes или других контейнеров.

```
VOLUME /path/to/volume
#Example
VOLUME /data
```

LABEL:

Добавляет метаданные к образу в виде пар ключ-значение.

```
LABEL key="value"
#Example
LABEL version="1.0" maintainer="Adrian"
```

USER:

Указывает имя пользователя или UID для использования при запуске образа.

```
USER user_name
#Example
USER app
```

ADD:

Копирует файлы или директории и может извлекать архивы в процессе сборки.

```
ADD source_path destination_path
#Example
ADD ./app.tar.gz /app
```

Похоже на COPY, но с дополнительными возможностями (например, извлечение архивов).

Пример

```
# Использовать официальный образ Node.js в качестве базового
FROM node:20-alpine

# Установить рабочую директорию в /app
WORKDIR /app

# Копировать package.json и package-lock.json в рабочую
директорию
COPY package*.json ./

# Установить зависимости
RUN npm install

# Копировать содержимое текущей директории в контейнер в /app
COPY . .

# Открыть порт 8080 для внешнего мира
EXPOSE 8080

# Определить переменную среды
ENV NODE_ENV=production

# Запустить app.js при старте контейнера
CMD node app.js
```

Шпаргалка по Docker Compose

Что такое файл Docker Compose?

Файл Docker Compose — это файл YAML, который определяет многоконтейнерное приложение Docker. Он указывает сервисы, сети и volumes для приложения, а также любые дополнительные опции конфигурации.

Синтаксис файла Docker Compose

version:

Указывает версию формата файла Docker Compose.

Пример:

```
version: '3.8'
```

services:

Определяет сервисы/контейнеры, которые составляют приложение.

Пример:

```
services:
   web:
      image: nginx:latest
```

networks:

Настроить пользовательские сети для приложения.

Пример:

```
networks:
   my_network:
      driver: bridge
```

command:

Переопределяет команду запуска по умолчанию, указанную в образе Docker.

Пример:

```
command: ["npm", "start"]
```

volumes:

Определяет именованные volumes, которые могут использоваться сервисами.

Пример:

```
volumes:
    my_volume:
```

environment:

Устанавливает переменные окружения для сервиса.

Пример:

```
environment:
    - NODE_ENV=production
```

ports:

Маппит порты хоста на порты контейнера. Т.е. определяет запросы на какой порт хост-машины будут перенаправляться на какой порт контейнера, запущенного на этой машине.

Пример:

```
ports:
    - "8080:80"
```

depends_on:

Указывает зависимости между сервисами, гарантируя, что один сервис запускается до другого.

Пример:

```
depends_on:
    - db
```

build:

Настройка контекста сборки и Dockerfile для сервиса.

Пример:

```
build:
   context: .
      dockerfile: Dockerfile.dev
```

volumes_from:

Подключает volumes от другого сервиса или контейнера.

Пример:

```
volumes_from:
    - service_name
```

Пример файла Docker Compose

Вот простой пример файла Docker Compose для веб-сервиса и базы данных:

```
version: '3.8'
# Определение сервисов для стека MERN
services:

  # Сервис MongoDB
   mongo:
      image: mongo:latest
      ports:
         - "27017:27017"
      volumes:
         - mongo_data:/data/db
      environment:
         MONGO_INITDB_ROOT_USERNAME:admin
         MONGO_INITDB_ROOT_PASSWORD:admin

# Node.js (Express) API сервис
api:
   build:
 # Указание контекста сборки для API сервиса
   context: ./api
```

```
    # Указание Dockerfile для сборки API сервиса
     dockerfile: Dockerfile

     ports:
      - "5000:5000"

# Обеспечение запуска сервиса MongoDB перед стартом API
  depends_on:
    - mongo

 environment:
    MONGO_URI: mongodb://admin:admin@mongo:27017/mydatabase

 networks:
    - mern_network

# React клиентский сервис
 client:
    build:
  # Указание контекста сборки для клиентского сервиса
    context: ./client
  # Указание Dockerfile для сборки клиентского сервиса

    dockerfile: Dockerfile

 ports:
    - "5000:5000"


# Убедиться, что сервис API запущен перед стартом клиента
    depends_on:
       - api

      networks:
        - mern_network

# Определить именованные volumes для постоянных данных
 volumes:
    mongo_data:

# Определить пользовательскую сеть для коммуникации между сервисами
 networks:
    mern_network:
```

Образы

Docker

1. Построение образа из Dockerfile:

```
> docker build -t image_name path_to_dockerfile
#Example
> docker build -t myapp .
```

2. Список всех локальных образов:

```
> docker images
#Example
> docker images ls
```

3. Скачивание образа из DockerHub:

```
> docker pull image_name:tag
#Example
> docker pull nginx:latest
```

4. Удаление локального образа:

```
> docker rmi image_name:tag
#Example
> docker rmi myapp:latest
```

Или

```
> docker rm [image_name/image_id]
#Example
> docker rm fd484f19954f
```

5. Добавление тега для образа:

```
> docker tag source_image:tag new_image:tag
#Example
> docker tag myapp:latest myapp:v1
```

6. Загрузить образ на DockerHub:

```
> docker push mage_name:tag
#Example
> docker push myapp:v1
```

7. Увидеть детали образа:

```
> docker image inspect image_name:tag
#Example
> docker image inspect myapp:v1
```

8. Сохранить образ в tar-архив:

```
> docker save -o image_name.tar image_name:tag
#Example
> docker save -o myapp.tar myapp:v1
```

9. Загрузить образ из tar-архива:

```
> docker load -i image_name.tar
#Example
> docker load -i myapp.tar
```

10. Удалить неиспользуемые образы:

```
> docker image prune
```

Docker

контейнер

1. Запустить контейнер на основе образа:

```
> docker run container_name image_name
#Example
> docker run myapp
```

2. Запустить именованный контейнер из образа:

```
> docker run --name container_name image_name:tag
#Example
> docker run --name my_container myapp:v1
```

3. Вывести список всех запущенных контейнеров:

```
> docker ps
```

4. Вывести список всех контейнеров (включая остановленные):

```
> docker ps -a
```

5. Остановить запущенный контейнер:

```
> docker stop container_name_or_id
#Example
> docker stop my_container
```

6. Запустить остановленный контейнер:

```
> docker start container_name_or_id
#Example
> docker start my_container
```

7. Запустить контейнер в интерактивном режиме:

```
> docker run -it container_name_or_id
#Example
> docker run -it my_container
```

8. Запустить контейнер в интерактивном режиме с shell:

```
> docker run -it container_name_or_id sh
#Example
> docker run -it my_container sh
```

9. Удалить остановленный контейнер:

```
> docker rm container_name_or_id
#Example
> docker rm my_container
```

10. Удалить запущенный контейнер (принудительно):

```
> docker rm -f container_name_or_id
#Example
> docker rm -f my_container
```

11. Посмотреть детали контейнера:

```
> docker inspect container_name_or_id
#Example
> docker inspect my_container
```

12. Просмотреть логи контейнера:

```
> docker logs container_name_or_id
#Example
> docker logs my_container
```

13. Приостановить работу запущенного контейнера:

```
> docker pause container_name_or_id
#Example
> docker pause my_container sh
```

14. Возобновить работу приостановленного контейнера:

```
> docker unpause container_name_or_id
#Example
> docker unpause my_container
```

Тома Docker и Сеть

1. Создать именованный volume:

```
> docker volume create volume_name
#Example
> docker volume create my_volume
```

2. Вывести список всех volumes:

```
> docker volume ls
```

3. Посмотреть детали volume:

```
> docker volume inspect volume_name
#Example
> docker volume inspect my_volume
```

4. Удалить volume:

```
> docker volume rm volume_name
#Example
> docker volume rm my_volume
```

5. Запустить контейнер с volume (mount):

```
> docker run --name container_name -v volume_name:/path/in/container
image_name:tag
#Example
> docker run --name my_container -v my_volume:/app/data myapp:v1
```

6. Копировать файлы между контейнером и volume:

```
> docker cp local_file_or_directory container_name:/path/in/container
#Example
> docker cp data.txt my_container:/app/data
```

Пример файла Docker Compose

1. Запустить контейнер с маршрутизацией портов:

```
> docker run --name container_name -p host_port:container_port image_name
#Example
> docker run --name container_name -p host_port:container_port image_name
```

2. Вывести список всех сетей:

```
> docker network ls
```

3. Посмотреть детали сети:

```
> docker network inspect network_name
#Example
> docker network inspect bridge
```

4. Создать пользовательскую мостовую сеть:

```
> docker network create network_name
#Example
> docker network create my_network
```

5. Подключить контейнер к сети:

```
> docker network connect network_name container_name
#Example
> docker network connect my_network my_container
```

6. Отключить контейнер от сети:

```
> docker network disconnect network_name container_name
#Example
> docker network disconnect my_network my_container
```

Docker Compose

1. Создать и запустить контейнеры, определенные в файле docker-compose.yml:

```
> docker compose up
```

Эта команда считывает файл docker-compose.yml и запускает определенные в нем сервисы в фоновом режиме в виде Docker-контейнеров.

2. Остановить и удалить контейнеры, определенные в файле docker-compose.yml:

```
> docker compose down
```

Эта команда останавливает и удаляет контейнеры, сети и volumes, определенные в файле docker-compose.yml.

3. Собрать или пересобрать сервисы:

```
> docker compose build
```

Эта команда собирает или пересобирает Docker образы для сервисов, определенных в файле docker-compose.yml.

4. Вывести список контейнеров для определенного проекта Docker Compose:

```
> docker compose ps
```

Эта команда выводит список контейнеров для сервисов, определенных в файле docker-compose.yml.

5. Просмотреть логи сервисов:

```
> docker compose logs
```

Эта команда показывает логи для всех сервисов, определенных в файле docker-compose.yml.

6. Масштабировать сервисы до определенного количества контейнеров:

```
> docker compose up -d --scale service_name=number_of_containers
#Example
> docker compose up -d --scale web=3
```

7. Выполнить одноразовую команду в сервисе:

```
> docker compose run service_name command
#Example
> docker compose run web npm install
```

8. Вывести список всех volumes:

```
> docker volume ls
```

Docker Compose создает volumes для сервисов. Эта команда помогает вам их видеть.

9. Приостановить сервис:

```
> docker compose pause service_name
```

Эта команда приостанавливает указанный сервис.

10. Возобновить работу сервиса:

```
> docker compose unpause service_name
```

Эта команда возобновляет работу указанного сервиса.

11. Просмотреть детали сервиса:

```
> docker compose ps service_name
```

Предоставляет подробную информацию о конкретном сервисе.

Последняя версия

Docker

1. Инициализировать Docker внутри приложения

```
> docker init
```

2. Отслеживать сервис/контейнер приложения

```
> docker compose watch
```

Он отслеживает контекст сборки для сервиса и перестраивает/обновляет контейнеры, когда файлы обновляются.

Надеемся, что руководство было полезно для тебя!

А если ты не хочешь останавливаться на достигнутом и намерен двигаться только вперед — ждем тебя на легендарном Java Буткемп!

Где под руководством действующих разработчиков из Яндекс и МТС за 5 месяцев ты освоишь навыки, необходимые современному IT-специалисту, а обучаясь и работая в команде с коллегами, приобретешь максимально приближенный к реальному опыт работы!

Программа Java Буткемп построена таким образом, что 90% обучения — практика, ты не просто прослушаешь теорию, а тут же применишь полученные знания и реализуешь фичи, о которых узнал!

Если остались вопросы, команда FAANG School с радостью ответит на них, для этого заполни форму ниже:

Записаться на БЕСПЛАТНУЮ консультацию