Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики, управления и технологий

ОТЧЕТ

по дисциплине «Интеграция и развертывание программного обеспечения с помощью контейнеров»
Лабораторная работа 2.1.: Создание Dockerfile и сборка образа
Направление подготовки – 38.03.05 «Бизнес-информатика».
профиль подготовки – «Аналитика данных и эффективное управление»

Выполнила:
студентка группы АДЭУ-211
st92
_____

Руководитель:

Кандидат технических наук, доцент
_____

Москва
2025 год

Цель работы: научиться создавать Dockerfile и собирать образы Docker для приложений.

Задачи:
Создать Dockerfile для указанного приложения.
Собрать образ Docker с использованием созданного Dockerfile.
Запустить контейнер из собранного образа и проверить его работоспособность.
Выполнить индивидуальное задание.

Ход работы:

Шаг 1: Написание простого Dockerfile
создадим новый каталог для нашего проекта:

```
dba@dba-vm:~$ mkdir my-python-app
dba@dba-vm:~$ cd my-python-app
dba@dba-vm:~/my-python-app$
```

Рисунок 1 – Выполнение команды для создания каталога

Создадим файл с именем app.py в каталоге my-python-app заполним кодом:

```
dba@dba-vm:~/my-python-app$ vi app.py
dba@dba-vm:~/my-python-app$ cat app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
        return "Hello, Docker!"

if __name__ == '__main__':
        app.run(host='0.0.0.0')
```

Рисунок 2 – Выполнение команды для создания файла app.py и просмотра содержимого

В том же каталоге создадим файл с именем Dockerfile (без расширения) и добавьте следующее содержимое:

```
dba@dba-vm:~/my-python-app$ vi Dockerfile
dba@dba-vm:~/my-python-app$ cat Dockerfile
# Используйте официальный образ Python из Docker Hub
FROM python:3.9-slim

# Установите рабочий каталог в контейнере
WORKDIR /app

# Скопируйте файл требований и установите зависимости
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Скопируйте код приложения в контейнер
COPY app.py app.py

# Откройте порт, на котором работает приложение
EXPOSE 5000

# Команда для запуска приложения
CMD ["python", "app.py"]
```

Рисунок 3 – Выполнение команды для создания Dockerfile

Создадим файл с именем requirements.txt в том же каталоге и добавьте следующую строку:

```
dba@dba-vm:~/my-python-app$ vi requirements.txt
dba@dba-vm:~/my-python-app$ cat requirements.txt
Flask
```

Рисунок 4 – Выполнение команды для создания requirements.txt

Шаг 2: Создайте свой образ Docker

В терминале перейдите в каталог my-python-app и выполните следующую команду для создания образа:

```
dba@dba-vm:~/my-python-app$ docker build -t my-python-app .
[+] Building 19.6s (10/10) FINISHED                                docker:default
 => [internal] load build definition from Dockerfile                         0.1s
 => => transferring dockerfile: 703B                                         0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim           2.0s
 => [internal] load .dockerignore                                            0.1s
```

Рисунок 5 – Выполнение команды создания докер образа

- Флаг -t помечает образ именем (my-python-app).
- . в конце указывает контекст сборки (текущий каталог).

После завершения процесса сборки проверьте, что образ был создан, запустив:

```
dba@dba-vm:~/my-python-app$ docker images
REPOSITORY      TAG       IMAGE ID       CREATED          SIZE
my-python-app   latest    88e9a0c91fa6   43 seconds ago   136MB
dba@dba-vm:~/my-python-app$
```

Рисунок 6 – Выполнение команды для просмотра образов

Шаг 3: Запустите ваш образ Docker

Используйте следующую команду для запуска вашего контейнера Docker:

```
dba@dba-vm:~/my-python-app$ docker run -p 5000:5000 my-python-app
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

Рисунок 7 – Выполнение команды для запуска приложения

Откроем веб-браузер и перейдите по адресу http://localhost:5000. Вы должны увидеть сообщение:

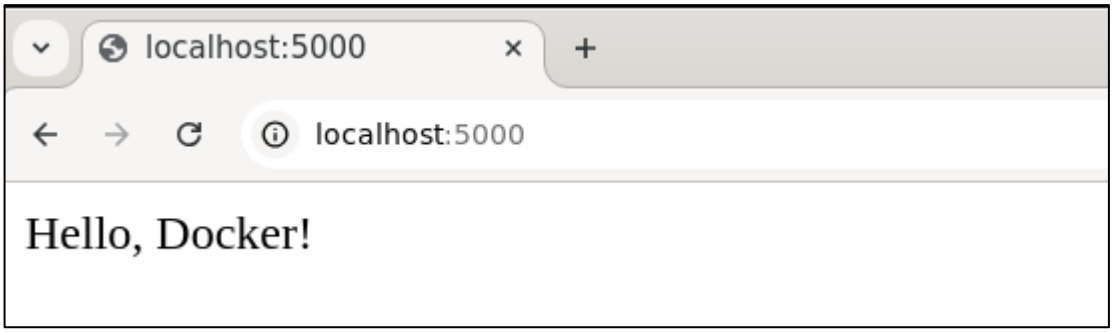

Рисунок 8 – Вывод по адресу localhost:5000

Шаг 4: Остановка контейнера

Чтобы остановить работающий контейнер, вы можете нажать Ctrl + C в терминале, где запущен контейнер, или открыть новый терминал и выполнить:

```
dba@dba-vm:~/my-python-app$ docker ps
CONTAINER ID   IMAGE           COMMAND           CREATED         STATUS
PORTS                                       NAMES
9045b409e8bc   my-python-app   "python app.py"   2 minutes ago   Up 2 minutes
0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp   gracious_mcclintock
dba@dba-vm:~/my-python-app$ docker stop 9045b409e8bc
9045b409e8bc
```

Рисунок 9 – Остановка контейнера docker

Выполним команды для удаления образа, очистки сети

```
dba@dba-vm:~/my-python-app$ sudo docker rm $(sudo docker ps -aq)
[sudo] password for dba:
9045b409e8bc
dba@dba-vm:~/my-python-app$ sudo docker rmi $(sudo docker images -q)
Untagged: my-python-app:latest
Deleted: sha256:110bc533b7a8562eb89c10fced24a798b86e6b420869ba305a279495ed2c245e
dba@dba-vm:~/my-python-app$ sudo docker system prune -a
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all images without at least one container associated to them
  - all build cache

Are you sure you want to continue? [y/N] y
Deleted build cache objects:
g9zjrtcl65tnzligt9zq7j967
srm3mu2oglr75lhdet5jo08hx
7wlyn3td1gcs9vm2q2sbdce5q
zhb9xm1fyzh0xc8gqlhn59014
f4th3sewpzq5c77qw0596dcok
```

Рисунок 10 – Выполнение команд удаления контейнера, образа и очистки сети

Шаг 5. Выполнение индивидуального задания

Вариант 5. Создайте Dockerfile для приложения на Ruby, которое выводит "Hello, Ruby!" при доступе к корневому URL.

Создадим каталог my-ruby-app

```
dba@dba-vm:~$ mkdir my-ruby-app
dba@dba-vm:~$ cd my-ruby-app
dba@dba-vm:~/my-ruby-app$
```

Рисунок 11 – Создание каталога

Создайте файл app.rb

```
dba@dba-vm:~/my-ruby-app$ vi app.rb
dba@dba-vm:~/my-ruby-app$ cat app.rb
require 'sinatra'

set :bind, '0.0.0.0'

get '/' do
  'Hello, Ruby!'
end
```

Рисунок 12 – Создание файла app с расширением rb

Создадим файл Gemfile, который описывает необходимые джемы

```
dba@dba-vm:~/my-ruby-app$ vi Gemfile
dba@dba-vm:~/my-ruby-app$ cat Gemfile
source 'https://rubygems.org'

gem 'sinatra'
gem 'rackup'
gem 'puma'
dba@dba-vm:~/my-ruby-app$
```

Рисунок 13 – Созданный Gemfile

Создадим Dockerfile

```
dba@dba-vm:~/my-ruby-app$ vi Dockerfile
dba@dba-vm:~/my-ruby-app$ cat Dockerfile
# Используем облегченный образ Ruby на базе Alpine
FROM ruby:3.0-alpine

# Устанавливаем зависимости для сборки Ruby-гемов и runtime-зависимости
RUN apk add --no-cache build-base gcompat

# Создаем рабочую директорию
WORKDIR /app

# Копируем Gemfile и устанавливаем зависимости
COPY Gemfile .
RUN bundle install --jobs $(nproc)

# Копируем основной файл приложения
COPY app.rb .

# Открываем порт, на котором работает Sinatra (по умолчанию 4567)
EXPOSE 4567

# Запускаем приложение с использованием Puma
CMD ["bundle", "exec", "ruby", "app.rb", "-s", "puma"]
```

Рисунок 14 – Созданный Dockerfile

Соберем docker - образ

```
dba@dba-vm:~/my-ruby-app$ docker build -t my-ruby-app .
[+] Building 25.9s (11/11) FINISHED                                    docker:default
 => [internal] load build definition from Dockerfile                           0.1s
 => => transferring dockerfile: 842B                                           0.0s
 => [internal] load metadata for docker.io/library/ruby:3.0-alpine             1.1s
 => [internal] load .dockerignore                                              0.1s
 => => transferring context: 2B                                                0.0s
 => [1/6] FROM docker.io/library/ruby:3.0-alpine@sha256:a59120aca45394ee979248e88667  3.8s
 => => resolve docker.io/library/ruby:3.0-alpine@sha256:a59120aca45394ee979248e88667  0.1s
 => => sha256:eb3556eca3c2b71460eabd894cbb7ae8c6e0c78b37db3b5b588f27 1.92kB / 1.92kB  0.0s
 => => sha256:0cc9e730d6fdcba4d20f68eb40b3c37ee9fe2b0f309550ff08457f 5.35kB / 5.35kB  0.0s
 => => sha256:a88dc8b54e91eb6b19695ef7e04865926d4df23004f414a3ee8697 2.81MB / 2.81MB  0.2s
 => => sha256:356551ca19a9c09baf21a8ee749d3fbb493608f7eb7a24dc2064bc 3.86MB / 3.86MB  0.4s
 => => sha256:df30d4881c233924e224a63696b00bec33b2d7f7bf93a3fa0ad6e5f066 194B / 194B  0.5s
 => => sha256:a59120aca45394ee979248e886672acaac91f175daf52cb91deb43 8.98kB / 8.98kB  0.0s
 => => extracting sha256:a88dc8b54e91eb6b19695ef7e04865926d4df23004f414a3ee869786174  0.2s
 => => sha256:9ca06a9f4d36d130cc0d53f48f766ec8fe8ebba03e9a50d36c52 26.23MB / 26.23MB  1.1s
 => => sha256:9640b58343af196871082fcc465e291757bfcfe297b141557706b0b119 140B / 140B  0.7s
 => => extracting sha256:356551ca19a9c09baf21a8ee749d3fbb493608f7eb7a24dc2064bcffef7  0.8s
 => => extracting sha256:df30d4881c233924e224a63696b00bec33b2d7f7bf93a3fa0ad6e5f0661  0.0s
 => => extracting sha256:9ca06a9f4d36d130cc0d53f48f766ec8fe8ebba03e9a50d36c528f3ec67  1.7s
 => => extracting sha256:9640b58343af196871082fcc465e291757bfcfe297b141557706b0b1196  0.0s
 => [internal] load build context                                              0.1s
 => => transferring context: 210B                                              0.0s
 => [2/6] RUN apk add --no-cache build-base gcompat                            7.2s
 => [3/6] WORKDIR /app                                                         0.2s
 => [4/6] COPY Gemfile .                                                       0.2s
 => [5/6] RUN bundle install --jobs $(nproc)                                  11.4s
 => [6/6] COPY app.rb .                                                        0.4s
 => exporting to image                                                         1.2s
 => => exporting layers                                                        1.1s
 => => writing image sha256:916d9c698e7d9ec3e64cd02209df94e8297a0adbb6a7dc294e9edce3  0.0s
 => => naming to docker.io/library/my-ruby-app                                 0.0s
```

Рисунок 15 – Сборка докер образа

Просмотрим созданный образ

```
dba@dba-vm:~/my-ruby-app$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
my-ruby-app   latest    b7cf3ca1ab3f   43 seconds ago   886MB
dba@dba-vm:~/my-ruby-app$
```

Рисунок 16 – Выполнение команды для просмотра образов

Запустим контейнер

```
dba@dba-vm:~/my-ruby-app$ docker run -p 4567:4567 my-ruby-app
== Sinatra (v4.1.1) has taken the stage on 4567 for development with backup from Puma
Puma starting in single mode...
* Puma version: 6.6.0 ("Return to Forever")
* Ruby version: ruby 3.0.7p220 (2024-04-23 revision 724a071175) [x86_64-linux-musl]
*  Min threads: 0
*  Max threads: 5
*  Environment: development
*          PID: 1
* Listening on http://0.0.0.0:4567
Use Ctrl-C to stop
172.17.0.1 - - [27/Feb/2025:21:23:43 +0000] "GET / HTTP/1.1" 200 12 0.0044
172.17.0.1 - - [27/Feb/2025:21:23:43 +0000] "GET /favicon.ico HTTP/1.1" 404 441 0.0019
```

Рисунок 17 – Выполнение команды для запуска контейнера

Перейдем по адресу localhost:4567 и убедимся, что приложение работает

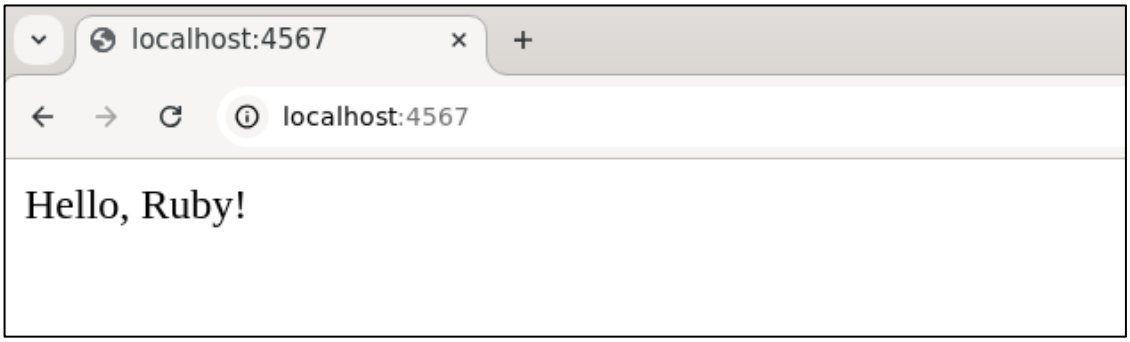

Рисунок 18 – Вывод по ссылке localhost:4567

Выводы:

В ходе работы мы научились создавать Dockerfile и собирать образы Docker для приложений. Задачи работы выполнены, а цель достигнута.

Контрольные вопросы:

1. Что такое Dockerfile и для чего он используется?
   Dockerfile — это текстовый файл, содержащий ряд инструкций по созданию образа Docker. Он определяет базовый образ, код приложения, зависимости и любые другие конфигурации, необходимые для создания образа.
2. Какие основные инструкции используются в Dockerfile?

Ключевые компоненты Dockerfile:

- FROM: указывает базовый образ для использования.
- COPY: копирует файлы из локальной файловой системы в образ.
- RUN: выполняет команды в образе во время процесса сборки.
- CMD: указывает команду для запуска при запуске контейнера из образа.

3. Как выполняется сборка образа Docker с использованием Dockerfile?

    1. Создается промежуточный контейнер на основе базового образа, указанного в FROM.
    2. Поочередно выполняются инструкции (компоненты) из Dockerfile
    3. После выполнения всех инструкций создается финальный образ, который сохраняется в локальной системе.

4. Как запустить контейнер из собранного образа?
   Используя команду docker run -p 5000:5000 myapp

5. Каковы преимущества использования Dockerfile для создания образов Docker?
    1. Процесс сборки образа полностью автоматизирован
    2. Позволяет создавать идентичные образы на разных системах(устройствах)
    3. Dockerfile можно хранить в системе контроля версий, что позволяет отслеживать изменения и возвращаться к предыдущим версиям