

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 3.1.: 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: освоить использование Docker Compose для управления многоконтейнерными приложениями.

Задачи:

1. Создать файл docker-compose.yml для указанного многоконтейнерного приложения.
2. Запустить приложение с помощью Docker Compose.
3. Проверить работоспособность приложения и взаимодействие между контейнерами.
4. Выполнить индивидуальное задание.

Ход работы:

Определим структуру:

-- flask-app

---app.py

---Dockerfile

---requirement.txt

---reviews.csv

--docker-compose.yml

Создать файл docker-compose.yml для указанного многоконтейнерного приложения

Шаг 1:Создадим app.py

```
dev@dev-vm:~/Lab3_1/flask-app$ cat app.py
from flask import Flask, request, jsonify
from elasticsearch import Elasticsearch
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
import nltk
from nltk.corpus import stopwords
import re

nltk.download('stopwords')

app = Flask(__name__)
es = Elasticsearch("http://elasticsearch-1:9200")

# Загрузка данных для обучения модели
data = pd.read_csv('reviews.csv') # Файл с отзывами для обучения
data['clean_text'] = data['text'].apply(lambda x: ' '.join([word for word in re.sub(r'[^\w\s]', '', x).lower().split() if word not in stopwords.words('english')]))

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(data['clean_text'])
y = data['sentiment']

model = MultinomialNB()
model.fit(X, y)

@app.route('/add_review', methods=['POST'])
def add_review():
    data = request.json
    text = data['text']
    clean_text = ' '.join([word for word in re.sub(r'[^\w\s]', '', text).lower().split() if word not in stopwords.words('english')])
    sentiment = model.predict(vectorizer.transform([clean_text]))[0]
```

Рисунок 1 – Содержимое app.py

Шаг 2: Создадим Dockerfile

```
dev@dev-vm: ~/lab3_1$ cat docker-compose.yml
# =====
# Platform Name: nosql-platform
# Platform Stack: trivadis/platys-modern-data-platform
# Platform Stack Version: develop
# =====

version: '3.8'
services:
  # ===== Elasticsearch ===== #
  elasticsearch-1:
    image: elasticsearch:8.13.0
    hostname: elasticsearch-1
    container_name: elasticsearch-1
    labels:
      com.platys.name: elasticsearch
      com.platys.description: Search-engine NoSQL store
      com.platys.restapi.title: Elasticsearch REST API
      com.platys.restapi.url: http://localhost:9200
      com.platys.manual.step.msgs: sudo sysctl -w vm.max_map_count=262144
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      discovery.type: single-node
      xpack.security.enabled: 'false'
      xpack.monitoring.collection.enabled: 'false'
      http.cors.enabled: 'true'
      http.cors.allow-origin: http://${DOCKER_HOST_IP}:28275,http://${PUBLIC_IP}:28275,http://dejavu:1358,http://localhost:28125,http://localhost:28125,http://${PUBLIC_IP}:28125,
R_HOST_IP:28125,http://127.0.0.1:1358
      http.cors.allow-headers: X-Requested-With,X-Auth-Token,Content-Type,Content-Length,Authorization
      http.cors.allow-credentials: 'true'
      cluster.routing.allocation.disk.threshold.enabled: 'true'
      cluster.routing.allocation.disk.watermark.low: 2gb
      cluster.routing.allocation.disk.watermark.high: 1gb
      cluster.routing.allocation.disk.watermark.flood_stage: F12gb
```

Рисунок 2 – Содержимое Dockerfile

Шаг 3: Создадим requirements.txt

```
dev@dev-vm: ~/lab3_1/flask-app$ cat requirements.txt
Flask
elasticsearch>=8.0.0
scikit-learn
nltk
pandas
```

Рисунок 3 – Содержимое requirements.txt

Шаг 4: Создадим docker-compose.yml

```

dev@dev-vm:~/lab3_1$ cat docker-compose.yml
# =====
# Platform Name          nosql-platform
# Platform Stack:        trivadis/platys-modern-data-platform
# Platform Stack Version: develop
# =====
# =====

version: '3.8'
services:
  # ===== Elasticsearch =====
  # ===== #
  elasticsearch-1:
    image: elasticsearch:8.13.0
    hostname: elasticsearch-1
    container_name: elasticsearch-1
    labels:
      com.platys.name: elasticsearch
      com.platys.description: Search-engine NoSQL store
      com.platys.restapi.title: Elasticsearch REST API
      com.platys.restapi.url: http://localhost:9200
      com.platys.manual.step.msgs: sudo sysctl -w vm.max_map_count=2
62144
    ports:
      - 9200:9200
      - 9300:9300
    environment:
      discovery.type: single-node
      xpack.security.enabled: 'false'
      xpack.monitoring.collection.enabled: 'false'
      http.cors.enabled: 'true'
      http.cors.allow-origin: http://${DOCKER_HOST_IP}:28275,http://
${PUBLIC_IP}:28275,http://dejavu:1358,http://localhost:28125,http://
localhost:28125,http://${PUBLIC_IP}:28125,http://${DOCKER_HOST_IP}:2
8125,http://127.0.0.1:1358
      http.cors.allow-headers: X-Requested-With,X-Auth-Token,Content
-Type,Content-Length,Authorization
      http.cors.allow-credentials: 'true'
      cluster.routing.allocation.disk.threshold_enabled: 'true'

```

Рисунок 4 – Содержимое docker-compose.yml

Структура приложения:

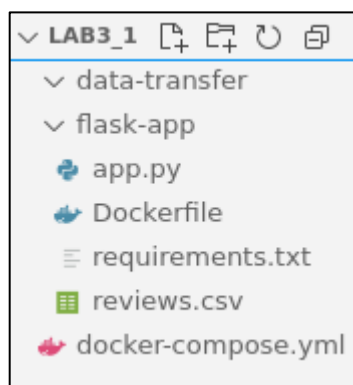


Рисунок 5 -- Структура приложения

Запустим приложение с помощью Docker Compose

Шаг 5: Выполним сборку образа и запустим контейнеры

```

dev@dev-vm:~/lab3_1$ docker-compose up --build
WARNING: The DOCKER_HOST_IP variable is not set. Defaulting to a blank string.
WARNING: The PUBLIC_IP variable is not set. Defaulting to a blank string.
Building flask-app
2025/03/07 15:48:21 in: []string{}
2025/03/07 15:48:21 Parsed entitlements: []
[+] Building 1.9s (2/2)
=> [internal] load build definition from Dockerfile                                docker:default 0.2s
=> => transferring dockerfile: 180B                                              0.1s
=> [internal] load metadata for docker.io/library/python:3.9                    1.5s

```

Рисунок 6 – Сборка и запуск контейнеров

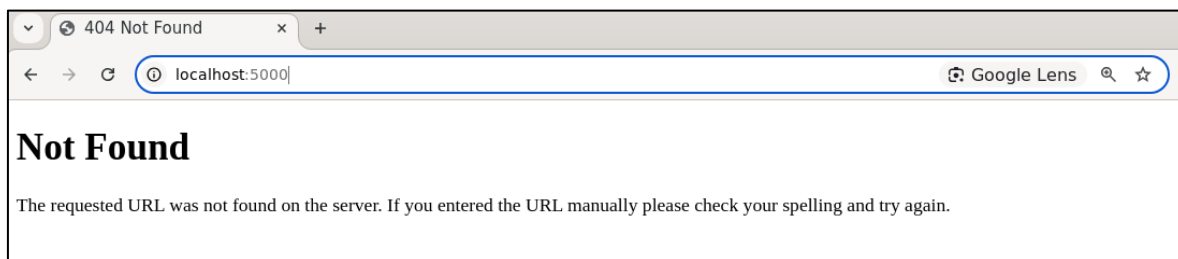


Рисунок 7 – Работоспособность по порту 5000

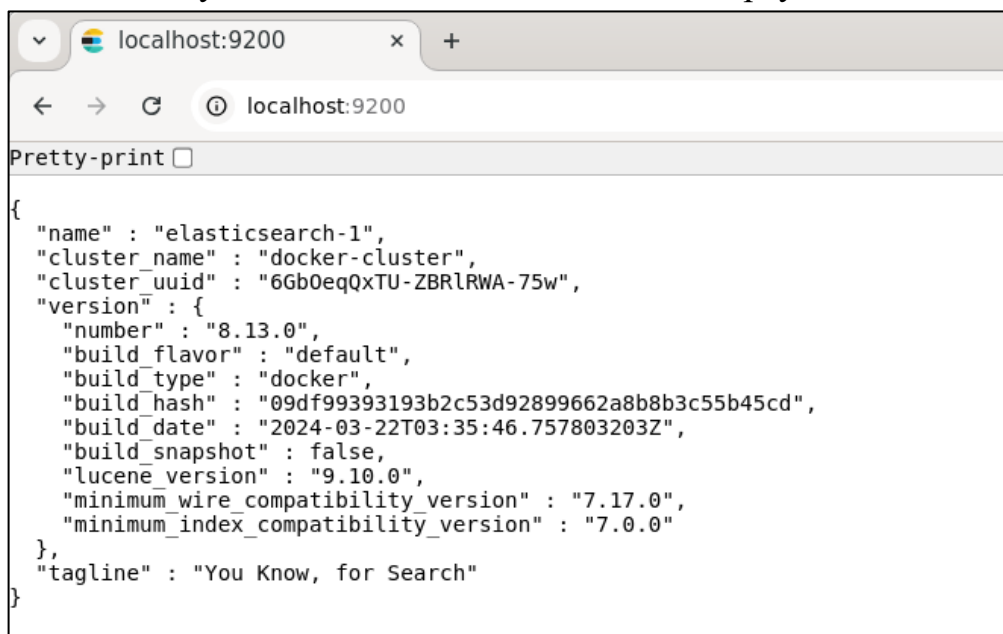


Рисунок 8 – Работоспособность по порту 9200

Опубликуем отзыв и запросим данные:

```

dev@dev-vm:~/lab3_1$ curl -X POST http://localhost:5000/add_review -H "Content-Type: application/json" -d '{"text": "This is a great product!"}'
{"message": "Review added", "sentiment": "positive", "status": "success"}
dev@dev-vm:~/lab3_1$ curl http://localhost:5000/search?q=great
[{"_id": "aRW0cJUB5nzDHoHPRqB0", "_index": "reviews", "_score": 0.2876821, "_source": {"sentiment": "positive", "text": "This is a great product!"}}]
dev@dev-vm:~/lab3_1$ cat docker-compose.yml

```

Рисунок 9 – Выполнение публикации и запрос по классификации

```

arch.cluster.uuid": "6Gb0eqQxTU-ZBRlRWA-75w", "elasticsearch.node.id": "-zb6nTq_SvmBUGZMPI2qWw", "elasticsearch.n
ode.name": "elasticsearch-1", "elasticsearch.cluster.name": "docker-cluster"}
elasticsearch-1 | {"@timestamp": "2025-03-07T12:22:07.274Z", "log.level": "INFO", "message": "[reviews/qeZ48
PvcSP2wxsQrPe0buw] create_mapping", "ecs.version": "1.2.0", "service.name": "ES ECS", "event.dataset": "elasticse
arch.server", "process.thread.name": "elasticsearch[elasticsearch-1][masterService#updateTask][T#18]", "log.logg
er": "org.elasticsearch.cluster.metadata.MetadataMappingService", "elasticsearch.cluster.uuid": "6Gb0eqQxTU-ZBRl
RWA-75w", "elasticsearch.node.id": "-zb6nTq_SvmBUGZMPI2qWw", "elasticsearch.node.name": "elasticsearch-1", "elasti
csearch.cluster.name": "docker-cluster"}
flask-app | 172.18.0.1 - - [07/Mar/2025 12:22:07] "POST /add_review HTTP/1.1" 200 -
flask-app | 172.18.0.1 - - [07/Mar/2025 12:22:16] "GET /search?q=great HTTP/1.1" 200 -
flask-app | 172.18.0.1 - - [07/Mar/2025 12:22:26] "GET /search?q=great HTTP/1.1" 200 -

```

Рисунок 10 – Ответ 200

Остановим приложение

```

...
flask-app | 172.18.0.1 - - [07/Mar/2025 12:41:16] code 400, message Bad request version ('jj\x13\x01
\x13\x02\x13\x03A+Ä/Ä,Ä0I@I"Ä\x13A\x14\x00\x9c\x00\x9d\x00/\x005\x01\x00\x06E\x1a\x1a\x00\x00\x
003\x04i\x04ië\x00\x01\x00\x11i\x04A\x17e&\x07'Ió\x088\x81.3KE3d8q;BAqzÇ\x15W0/ s\x84à\x93k?«;Ü\x14\x0b\x1
4AF2;j·'Z\x1bjH&öw"\x87" HTTPStatus.BAD_REQUEST -
flask-app | 172.18.0.1 - - [07/Mar/2025 12:41:16] "GET / HTTP/1.1" 404 -
flask-app | 172.18.0.1 - - [07/Mar/2025 12:41:19] "GET /favicon.ico HTTP/1.1" 404 -
^CGracefully stopping... (press Ctrl+C again to force)
Stopping flask-app ...
Stopping cerebro ... done
Stopping dejavu ...
Stopping elasticsearch-1 ...
Stopping elasticsearch-1 ...

```

Рисунок 11 – Остановка приложения

Выводы:

В ходе работы мы освоили использование Docker Compose для управления многоконтейнерными приложениями

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?
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?
Сборка образа Docker выполняется с помощью команды `docker build`
4. Как запустить контейнер из собранного образа?
Для запуска контейнера из собранного образа используется команда `docker run`

5. Каковы преимущества использования Dockerfile для создания образов Docker?

- Dockerfile позволяет создавать идентичные образы на разных машинах, что упрощает развертывание и тестирование.
- Процесс сборки образа автоматизирован, что снижает вероятность ошибок, связанных с ручной настройкой.
- Dockerfile можно хранить в системе контроля версий (например, Git), что позволяет отслеживать изменения и возвращаться к предыдущим версиям.
- Dockerfile позволяет использовать базовые образы и добавлять только необходимые изменения, что упрощает управление зависимостями.
- Dockerfile явно описывает, что именно будет содержаться в образе, что делает процесс сборки более понятным и контролируемым.
- Dockerfile можно использовать для создания множества образов с разными настройками, что экономит время и ресурсы.