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

ДИСЦИПЛИНА:

Интеграция и развертывание программного обеспечения с помощью контейнеров

Лабораторная работа 3.1

Docker Compose для мультиконтейнерных приложений

Выполнила: st_105

Москва
2025

Цель работы: освоить использование Docker Compose для управления многоконтейнерными приложениями.

Задачи:

— Создать файл docker-compose.yml для указанного многоконтейнерного приложения.

— Запустить приложение с помощью Docker Compose.

— Проверить работоспособность приложения и взаимодействие между контейнерами.

— Выполнить индивидуальное задание.

ХОД РАБОТЫ:

1. Создана директория st_105 для дальнейшей работы

   Директория templates создана для файлов .html

```
dev@dev-vm:~$ mkdir st_105
dev@dev-vm:~$ cd st_105
dev@dev-vm:~/st_105$ mkdir templates
dev@dev-vm:~/st_105$ ls
templates
```

2. Создан файл docker-compose.yml

(Dockerfile — для сборки образов.

docker-compose.yml — для управления многоконтейнерными приложениями.)

```
version: '3.8'
services:
  web:
    image: python:3.9-slim
    container_name: flask-app
    working_dir: /app
    volumes:
      - .:/app
    ports:
      - "5000:5000"
    command: sh -c "pip install -r requirements.txt && python app.py"  # Установка зависимостей и запуск приложения
    depends_on:
      - influxdb
    environment:
      INFLUXDB_URL: http://influxdb:8086
      INFLUXDB_TOKEN: my-super-secret-token
      INFLUXDB_ORG: my-org
      INFLUXDB_BUCKET: my-bucket

  influxdb:
```

```
    image: influxdb:2.0
    container_name: influxdb
    ports:
      - "8086:8086"
    volumes:
      - influxdb-data:/var/lib/influxdb2
    environment:
      DOCKER_INFLUXDB_INIT_MODE: setup
      DOCKER_INFLUXDB_INIT_USERNAME: admin
      DOCKER_INFLUXDB_INIT_PASSWORD: admin123
      DOCKER_INFLUXDB_INIT_ORG: my-org
      DOCKER_INFLUXDB_INIT_BUCKET: my-bucket
      DOCKER_INFLUXDB_INIT_ADMIN_TOKEN: my-super-secret-token

volumes:
  influxdb-data:
```

Файл описывает конфигурацию для запуска многоконтейнерного приложения, состоящего из двух сервисов: Flask-приложения (веб-сервер) и InfluxDB (база данных временных рядов).

3. Далее создан файл app.py где расписана логика Flask приложения

```
from flask import Flask, render_template, request
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS
import pandas as pd
from sklearn.linear_model import LinearRegression
from datetime import datetime, timedelta

app = Flask(__name__)

# Конфигурация InfluxDB
INFLUXDB_URL = "http://influxdb:8086"
INFLUXDB_TOKEN = "my-super-secret-token"
INFLUXDB_ORG = "my-org"
INFLUXDB_BUCKET = "my-bucket"

client = InfluxDBClient(url=INFLUXDB_URL, token=INFLUXDB_TOKEN,
org=INFLUXDB_ORG)
write_api = client.write_api(write_options=SYNCHRONOUS)
query_api = client.query_api()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/record', methods=['POST'])
def record_visit():
    # Запись данных о посещаемости
    point = Point("visits").tag("location", "main").field("count", 1)
    write_api.write(bucket=INFLUXDB_BUCKET, record=point)
    return "Visit recorded!"

@app.route('/analytics')
def analytics():
    # Получение данных о посещаемости
    query = f'
    from(bucket: "{INFLUXDB_BUCKET}")
      |> range(start: -30d)
```

```
    |> filter(fn: (r) => r._measurement == "visits")
    |> aggregateWindow(every: 1d, fn: sum, createEmpty: false)
  '''
  result = query_api.query(query)

  # Преобразование данных в DataFrame
  data = []
  for table in result:
    for record in table.records:
      data.append({
        "time": record.get_time(),
        "count": record.get_value()
      })
  df = pd.DataFrame(data)

  # Прогнозирование посещаемости на следующий месяц
  if not df.empty:
    df['time'] = pd.to_datetime(df['time'])
    df['days'] = (df['time'] - df['time'].min()).dt.days
    model = LinearRegression()
    model.fit(df[['days']], df['count'])
    future_days = df['days'].max() + 30
    predicted_count = model.predict([[future_days]])[0]
  else:
    predicted_count = 0

  return render_template('analytics.html', visits=df.to_dict('records'),
predicted_count=int(predicted_count))

if __name__ == '__main__':
  app.run(host='0.0.0.0', port=5000)
```

- Содержит логику Flask-приложения:
  - Запись данных о посещаемости в InfluxDB.
  - Получение данных из InfluxDB и их анализ.
  - Прогнозирование посещаемости на следующий месяц.
- Отображает данные через HTML-шаблоны.

4. Далее создан файл зависимостей requirements.txt

```
Flask
influxdb-client
pandas
scikit-learn
```

5. Создание html страниц: главная страница,где пользователь может записать посещение и страница аналитики

Index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Аналитика посещаемости</title>
</head>
<body>
  <h1>Аналитика посещаемости</h1>
  <h2>Посещения за последние 30 дней:</h2>
  <ul>
```

```
        {% for visit in visits %}
        <li>{{ visit.time }}: {{ visit.count }} посещений</li>
        {% endfor %}
    </ul>
    <h2>Прогноз посещаемости на следующий месяц: {{ predicted_count }} посещений</h2>
    <a href="/">Вернуться на главную</a>
</body>
</html>
```

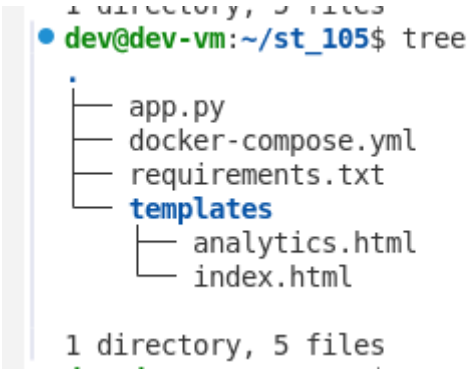
- Отображает кнопку для записи посещения.
- Содержит ссылку на страницу аналитики.

analytics.html

Страница аналитики, где отображаются данные о посещаемости и прогноз на следующий месяц.

```
<!DOCTYPE html>
<html>
<head>
    <title>Аналитика посещаемости</title>
</head>
<body>
    <h1>Аналитика посещаемости</h1>
    <h2>Посещения за последние 30 дней:</h2>
    <ul>
        {% for visit in visits %}
        <li>{{ visit.time }}: {{ visit.count }} посещений</li>
        {% endfor %}
    </ul>
    <h2>Прогноз посещаемости на следующий месяц: {{ predicted_count }} посещений</h2>
    <a href="/">Вернуться на главную</a>
</body>
</html>
```

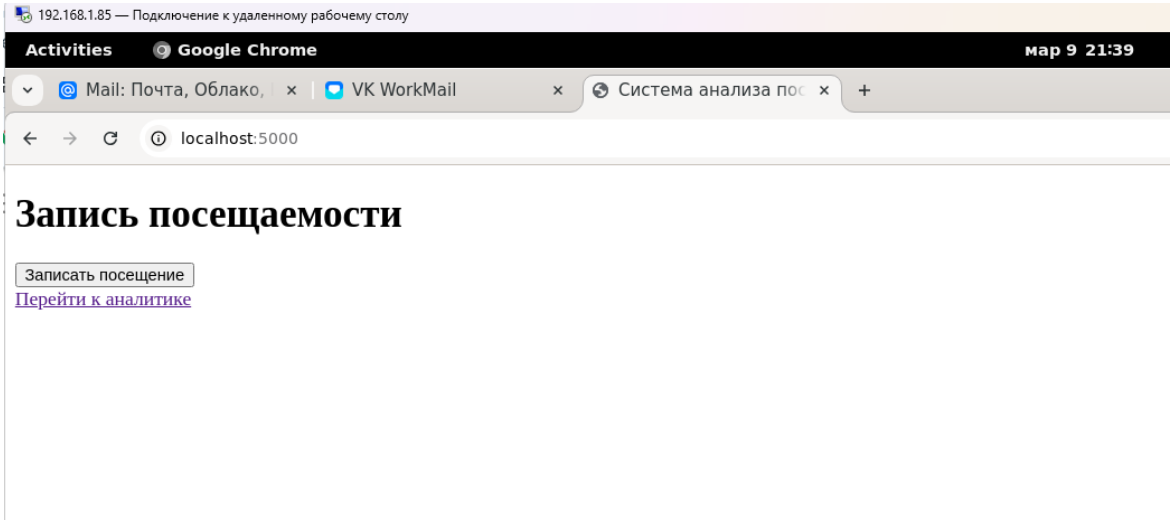
Итоговая структура файлов:



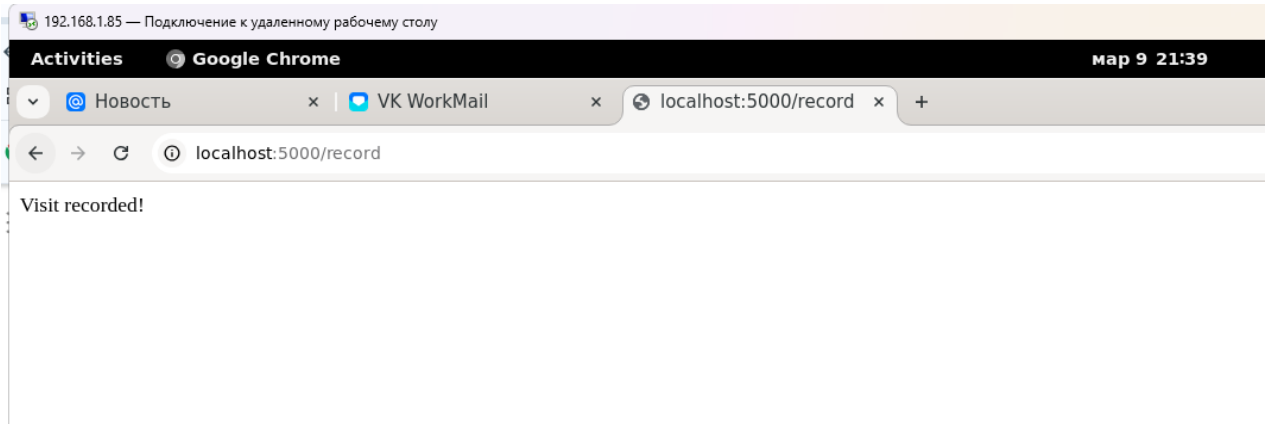
Запуск контейнера с помощью **docker compose up --build**:

```
dev@dev-vm:~/st_105$ docker compose up --build
WARN[0000] /home/dev/st_105/docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 3/3
 ✔ Network st_105_default  Created
 ✔ Container influxdb      Created
 ✔ Container flask-app     Created
Attaching to flask-app, influxdb
influxdb   | time="2025-03-09T18:28:13Z" level=info msg="reset OCSP cache file. /root/.cache/snowflake/ocsp_response_cache.json" func="gosnowflake.(*defaultLogger).Infof
influxdb   | time="2025-03-09T18:28:13Z" level=info msg="reading OCSP Response cache file. /root/.cache/snowflake/ocsp_response_cache.json\n" func="gosnowflake.(*default
influxdb   | time="2025-03-09T18:28:13Z" level=error msg="failed to open. Ignored. open /root/.cache/snowflake/ocsp_response_cache.json: no such file or directory\n" fun
" file="log.go:120"
influxdb   | time="2025-03-09T18:28:13Z" level=info msg="reset OCSP cache file. /root/.cache/snowflake/ocsp_response_cache.json" func="gosnowflake.(*defaultLogger).Infof
influxdb   | time="2025-03-09T18:28:13Z" level=info msg="reading OCSP Response cache file. /root/.cache/snowflake/ocsp_response_cache.json\n" func="gosnowflake.(*default
influxdb   | time="2025-03-09T18:28:13Z" level=error msg="failed to open. Ignored. open /root/.cache/snowflake/ocsp_response_cache.json: no such file or directory\n" fun
" file="log.go:120"
influxdb   | time="2025-03-09T18:28:14Z" level=info msg="reset OCSP cache file. /root/.cache/snowflake/ocsp_response_cache.json" func="gosnowflake.(*defaultLogger).Infof
influxdb   | time="2025-03-09T18:28:14Z" level=info msg="reading OCSP Response cache file. /root/.cache/snowflake/ocsp_response_cache.json\n" func="gosnowflake.(*default
influxdb   | time="2025-03-09T18:28:14Z" level=error msg="failed to open. Ignored. open /root/.cache/snowflake/ocsp_response_cache.json: no such file or directory\n" fun
" file="log.go:120"
influxdb   | time="2025-03-09T18:28:14Z" level=info msg="reset OCSP cache file. /home/influxdb/.cache/snowflake/ocsp_response_cache.json" func="gosnowflake.(*defaultLogg
influxdb   | time="2025-03-09T18:28:14Z" level=info msg="reading OCSP Response cache file. /home/influxdb/.cache/snowflake/ocsp_response_cache.json\n" func="gosnowflake.
04"
influxdb   | time="2025-03-09T18:28:14Z" level=error msg="failed to open. Ignored. open /home/influxdb/.cache/snowflake/ocsp_response_cache.json: no such file or directo
r).Errorf" file="log.go:120"
```

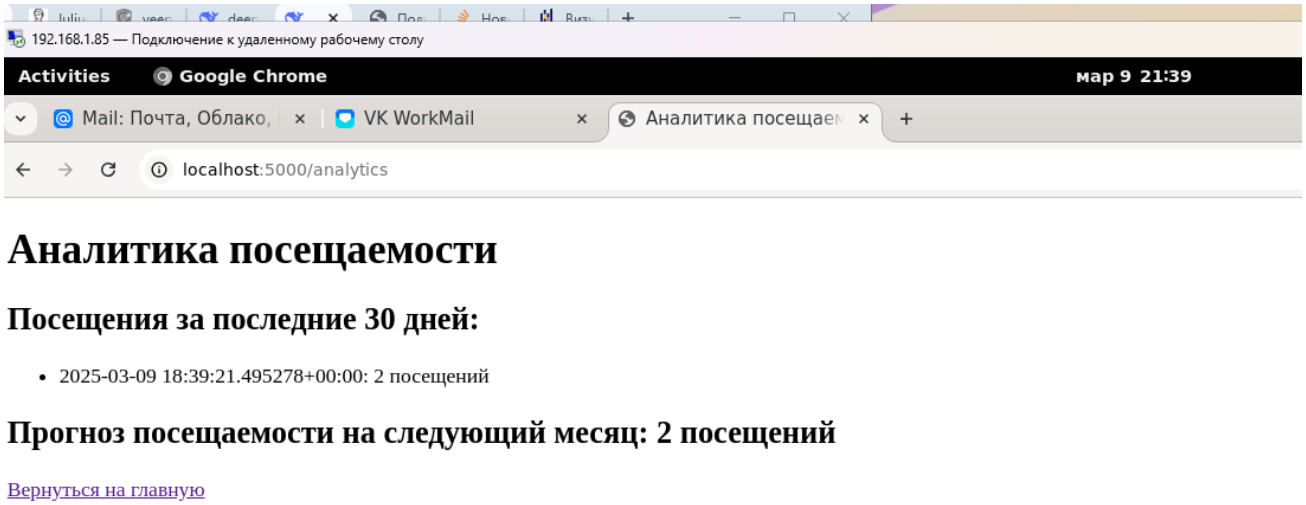
Localhost:5000



Localhost:5000/record



Localhost:5000/analytics



Выводы: В ходе работы было создано многоконтейнерное приложение для анализа посещаемости с использованием Docker Compose, Flask и InfluxDB. Реализованы функции записи данных, их анализа и прогнозирования посещаемости на следующий месяц. Приложение успешно запускается в контейнерах, взаимодействует с базой данных и отображает результаты через веб-интерфейс.