

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

**Отчет**

**Лабораторная работа №4**

Выполнил:  
Кривцов Павел  
Группа К33402

Проверил:  
Добряков Д. И.

Санкт-Петербург  
2023 г.

## Задача

Необходимо упаковать приложение в docker-контейнеры и обеспечить сетевое взаимодействие между различными его частями.

## Ход работы

Логика работы микросервисов оставим без изменений и напишем к каждому по докерфайлу. Выглядеть он будет следующим образом и отличаться только портами для каждого микросервиса:

```
1  ➤ FROM node:19.8.1-alpine
2
3  WORKDIR /gateway
4
5  COPY package*.json ./
6
7  RUN npm install
8
9  COPY . .
10
11 EXPOSE 8000
12
13 CMD [ "npm", "start" ]
```

Файл docker-compose будет выглядеть так (видно, что есть сущность gateway, зависящая от users и kinopoisk):

```
1   version: '3.9'
2
3  >> services:
4  > gateway:
5      container_name: gateway
6      build:
7          context: ./gateway
8      depends_on:
9          - users
10         - kinopoisk
11      ports:
12          - "127.0.0.1:8000:8000"
13      restart: always
14  > users:
15      container_name: users
16      build:
17          context: ./users
18      ports:
19          - "8001"
20      restart: always
21  > kinopoisk:
22      container_name: kinopoisk
23      build:
24          context: ./kinopoisk
25      ports:
26          - "8002"
27      restart: always
28
```

Немного модернизируем переадресацию в index.ts файле гейтвэя, чтобы вместо localhost указывалось название докер-контейнера, к которому направлен запрос.

```
18
19 for (const m in microservices) {
20   app.all( path: `/${m}/*`, handlers: async (req: any, res: any) => {
21     const url = `http://${m}:${microservices[m]}${req.originalUrl}`;
22     try {
23       const response = await axios({
24         method: req.method,
25         url: url,
26         data: req.body,
27         headers: {
28           auth: req.headers.auth
29         }
30       });
31       res.status(response.status).send(response.data);
32     } catch (e) {
33       if (e.response) {
34         res.status(e.response.status).send(e.response.data);
35       } else {
36         res.status(500).send('Internal Server Error');
37       }
38     }
39   });
40 }
```

В результате запускается 3 докер-контейнера, которые общаются с внешним миром через gateway на 8000-м порту.

Container CPU usage ⓘ  
0.00% / 400% (4 cores allocated)

Container memory usage ⓘ  
631.2MB / 3.75GB

Show charts ▾

⏸





☐ Only show running containers

Delete

▶

⏸

■

<input checked="" type="checkbox"/>	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input checked="" type="checkbox"/>	 lw4		Running (3/3)	0%		3 hours ago	<div>■ ⋮ 🗑</div>
<input checked="" type="checkbox"/>	 gateway	lw4-gateway	Running	0%	8000:8000 <a href="#">🔗</a>	3 hours ago	<div>■ ⋮ 🗑</div>
<input checked="" type="checkbox"/>	 kinopoisk	lw4-kinopoisk	Running	0%	64531:8002 <a href="#">🔗</a>	3 hours ago	<div>■ ⋮ 🗑</div>
<input checked="" type="checkbox"/>	 users	lw4-users	Running	0%	64532:8001 <a href="#">🔗</a>	3 hours ago	<div>■ ⋮ 🗑</div>

## **Вывод**

В ходе работы удалось создать docker-compose и наладить сетевое взаимодействие между разными docker-контейнерами (микросервисами) приложения. Документация к приложению:

<https://documenter.getpostman.com/view/29513351/2s9YBz3b3P>