САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 4

Выполнил:

Беломытцев Андрей

К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Контейнеризация написанного приложения средствами docker

- реализовать Dockerfile для каждого сервиса;
- написать общий docker-compose.yml;
- настроить сетевое взаимодействие между сервисами.

Ход работы

Написаны файлы Dockerfile для каждого из трёх микросервисов. Вот Dockerfile для user-service, для других микросервисов аналогично

```
FROM node:20 AS builder

WORKDIR /var/www/apps/user
COPY . /var/www/apps/user

RUN npm install --omit=optional

RUN npm run build

FROM node:20 AS prod

WORKDIR /var/www/apps/user
COPY --from=builder /var/www/apps/user/dist /var/www/apps/user
COPY --from=builder /var/www/apps/user/node_modules
/var/www/apps/user/node_modules

CMD node /var/www/apps/user/index.js
```

Написан файл docker-compose.yml с тремя сервисами, с тремя PostgreSQL и с RabbitMQ

```
services:
  rabbitmq:
    image: rabbitmq:management
    ports:
      - '5672:5672'
      - '15672:15672'
    environment:
      RABBITMQ_DEFAULT_USER: ${RABBITMQ_DEFAULT_USER}
      RABBITMQ_DEFAULT_PASS: ${RABBITMQ_DEFAULT_PASS}
    networks:
      - backend
    healthcheck:
      test: rabbitmq-diagnostics -q ping
      interval: 30s
      timeout: 30s
      retries: 3

  user-db:
    image: postgres
    container_name: user-db
```

```
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
      PGPORT: ${DB_PORT_USER}
    ports:
      - "${DB_PORT_USER}:${DB_PORT_USER}"
    volumes:
      - user-db:/var/lib/postgresql/data
    networks:
      - backend
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -d $${POSTGRES_DB} -U
$${POSTGRES_USER}"]
      interval: 10s
      timeout: 5s
      retries: 5

  channel-db:
    image: postgres
    container_name: channel-db
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
      PGPORT: ${DB_PORT_CHANNEL}
    ports:
      - "${DB_PORT_CHANNEL}:${DB_PORT_CHANNEL}"
    volumes:
      - channel-db:/var/lib/postgresql/data
    networks:
      - backend
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -d $${POSTGRES_DB} -U
$${POSTGRES_USER}"]
      interval: 10s
      timeout: 5s
      retries: 5

  video-db:
    image: postgres
    container_name: video-db
    restart: unless-stopped
    environment:
      POSTGRES_USER: ${DB_USER}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_NAME}
      PGPORT: ${DB_PORT_VIDEO}
    ports:
      - "${DB_PORT_VIDEO}:${DB_PORT_VIDEO}"
    volumes:
      - video-db:/var/lib/postgresql/data
    networks:
      - backend
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -d $${POSTGRES_DB} -U
$${POSTGRES_USER}"]
      interval: 10s
      timeout: 5s
```

```
      retries: 5

  user-service:
    container_name: user-service
    build:
      context: ./user-service
      dockerfile: Dockerfile
    depends_on:
      user-db:
        condition: service_healthy
      rabbitmq:
        condition: service_healthy
    env_file:
      - .env
    ports:
      - '3000:3000'
    networks:
      - backend

  channel-service:
    container_name: channel-service
    build:
      context: ./channel-service
      dockerfile: Dockerfile
    depends_on:
      channel-db:
        condition: service_healthy
      rabbitmq:
        condition: service_healthy
    env_file:
      - .env
    ports:
      - '3001:3000'
    networks:
      - backend

  video-service:
    container_name: video-service
    build:
      context: ./video-service
      dockerfile: Dockerfile
    depends_on:
      video-db:
        condition: service_healthy
      rabbitmq:
        condition: service_healthy
    env_file:
      - .env
    ports:
      - '3002:3000'
    networks:
      - backend

volumes:
  user-db:
  channel-db:
  video-db:

networks:
  backend:
```

Сетевое взаимодействие реализовано с помощью RabbitMQ.

В микросервисах создан файл rabbit.ts с функциями sendToQueue и listenToQueue используемыми для взаимодействия с RabbitMQ с использованием amqplib.

```
import amqplib from 'amqplib'

export const sendToQueue = async (queue: string, message: any) => {
  const conn = await amqplib.connect('amqp://rabbitmq')
  const channel = await conn.createChannel()
  await channel.assertQueue(queue, { durable: true })
  channel.sendToQueue(queue, Buffer.from(JSON.stringify(message)))
  console.log(`Message sent to ${queue}:`, message)
  await channel.close()
  await conn.close()
}

export const listenToQueue = async (queue: string, callback: (content: any)
=> any) => {
  const conn = await amqplib.connect('amqp://rabbitmq')
  const channel = await conn.createChannel()
  await channel.assertQueue(queue, { durable: true })
  channel.consume(queue, (msg) => {
    if (msg) {
      const content = JSON.parse(msg.content.toString())
      console.log(`Received from ${queue}:`, content)
      callback(content)
      channel.ack(msg)
    }
  })
}
```

Реализована передача сообщений от channel-service к video-service. В моменты добавления и удаления канала, соотвествующие видео должны добавляться и удаляться соотвественно. Реализуя нечто вроде cascade, который использовался, когда API ещё было монолитным.

Следующий код добавлен в channel-service

```
import { sendToQueue, listenToQueue } from '../rabbit'
```

Запускается при добавлении канала

```
await sendToQueue('add_videos', { channelId: channelId })
```

Запускается при удалении канала

```
await sendToQueue('delete_videos', { channelId: id })
```

Следующий код добавлен в video-service

```
import { sendToQueue, listenToQueue } from './rabbit'

const repository = AppDataSource.getRepository(Video)

const getVideos = async (channelId: string, maxResults: number = 50) => {
  const uploads = 'UULF' + channelId.slice(2)
  const videos: any = await (await
fetch(`https://www.googleapis.com/youtube/v3/playlistItems?part=snippet%2Ccon
tentDetails&maxResults=${maxResults}&playlistId=${uploads}&key=${config.YT_AP
I_KEY}`)).json()
  const videosList: Video[] = []
  for(let m of videos['items']){
    m = m['snippet']
    videosList.push({
      'id': m['resourceId']['videoId'],
      'channelId': m['channelId'],
      'title': m['title'],
      'publishedAt': m['publishedAt'],
      'thumbnail': m['thumbnails']['maxres' in m['thumbnails'] ? 'maxres' :
'medium']['url'],
      'description': m['description'],
    } as Video)
  }
  return repository.save(videosList)
}

const deleteVideos = async (channelId: string) => {
  await repository.delete({ channelId: channelId })
}

listenToQueue('add_videos', (content) => getVideos(content.channelId))
listenToQueue('delete_videos', (content) => deleteVideos(content.channelId))
```

Страницы с документациями (http://127.0.0.1:3000/docs/, http://127.0.0.1:3001/docs/, http://127.0.0.1:3002/docs/) работают, как и должны.

Протестирована работа API с помощью расширения REST Client для VS Code. Для проверки взаимодействия всех микросервисов между собой проведены следующие тесты:

Регистрация

```
POST http://127.0.0.1:3000/user/register
Content-Type: application/json

{
  "username": "andrei",
  "email": "andrei@example.com",
  "password": "qwerty"
}
```

Получение JWT

```
POST http://127.0.0.1:3000/user/login
```

```
Content-Type: application/json

{
  "username": "andrei",
  "password": "qwerty"
}
```

Добавление канала

```
POST http://127.0.0.1:3001/channel
Authorization: Bearer ...
Content-Type: application/json

{
  "id": "UCHnyfMqiRRG1u-2MsSQLbXA",
  "lang": "en",
  "category": "popsci",
  "theme": "all"
}
```

Проверка появился ли канал (да)

```
GET http://127.0.0.1:3001/channel
```

Проверка появились ли видео (да)

```
GET http://127.0.0.1:3002/video
```

Удаление видео (проверено, что истёкший JWT или JWT без нужных прав не работает)

```
DELETE http://127.0.0.1:3001/channel/UCHnyfMqiRRG1u-2MsSQLbXA
Authorization: Bearer ...
Content-Type: application/json
```

Проверка пропал ли канал (да)

```
GET http://127.0.0.1:3001/channel
```

Проверка пропали ли видео (да)

```
GET http://127.0.0.1:3002/video
```

Вывод

В результате была реализована контейнеризация написанного приложения средствами docker. Реализованы файлы Dockerfile для каждого из трёх сервисов. Написан общий docker-compose.yml с тремя сервисами, с тремя PostgreSQL и с RabbitMQ. Настроено сетевое взаимодействие между сервисами с помощью RabbitMQ. Протестирована работа API.