САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 4

Выполнил:
Байков Иван
К33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Необходимо упаковать ваше приложение в docker-контейнеры и обеспечить сетевое взаимодействие между различными частями вашего приложения, а также настроить общение микросервисов между собой посредством RabbitMQ. Делать это можно как с помощью docker-compose так и с помощью docker swarm.

Ход работы

1) Для начала подключим RabbitMq на обоих сервисах:

```
import amqplib from "amqplib";
import "dotenv/config";

export let connection: amqplib.Connection;
export let channel: amqplib.Channel;
export const requestQueue = "auth-req";
export default async function connect() {
  try {
    console.log("RabbitMQ connected: server");
    connection = await amqplib.connect(process.env.AMQP);
    channel = await connection.createChannel();
    await channel.assertQueue(requestQueue);
  } catch (error) {
    console.log(error);
  }
}
```

```
import amqplib from "amqplib";
import "dotenv/config";
import { listen } from "./auth.js";

export let connection: amqplib.Connection;
export let channel: amqplib.Channel;
export const requestQueue = "auth-req";
export default async function connect() {
  try {
    console.log("RabbitMQ connected: auth");
    connection = await amqplib.connect(process.env.AMQP);
    channel = await connection.createChannel();
    await channel.assertQueue(requestQueue);
    listen();
  } catch (error) {
    console.log(error);
  }
}
```

2)

```
import { Request, Response, NextFunction } from "express";
import { channel, requestQueue } from "../rabbitmq/connect.js";
import crypto from "crypto";

const authMiddleware = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const authHeader = req.headers.authorization;

    if (!authHeader || !authHeader.startsWith("Bearer")) {
      return res.status(401).json({ message: "No token provided" });
    }

    const token = authHeader.split(" ")[1];
    const correlationId = crypto.randomUUID();

    const q = await channel.assertQueue(`${token}-${correlationId}`, { exclusive: true });

    channel.sendToQueue(requestQueue, Buffer.from(token), {
      correlationId: correlationId,
      replyTo: q.queue,
    });

    channel.consume(
      q.queue,
      (msg) => {
        if (msg.properties.correlationId === correlationId) {
          console.log("Response", msg.content.toString());
          const response = JSON.parse(msg.content.toString());

          if (response.valid) {
            // @ts-ignore
            req.user = response.user;
            next();
          } else {
            res.status(401).json({ message: "Invalid token" });
          }
          channel.cancel(q.queue);
        }
      },
      { noAck: true }
    );
  } catch (error) {
    console.log(error);
    res.status(500).json("Internal server error");
  }
};

export default authMiddleware;
```

3) В auth сервисе напишем consumer

```
export async function listen() {
Click to collapse the range. requestQueue, (msg) => {
    const token = msg.content.toString();
    console.log("Received token:", token);
    let response = {
      valid: false,
      user: null,
    };

    // Validate the token
    try {
      const user = jwt.verify(token, process.env.JWT_SECRET);
      response.valid = true;
      response.user = user;
    } catch (err) {
      response.valid = false;
    }

    // Send back the validation result to the server service
    const sent = channel.sendToQueue(msg.properties.replyTo, Buffer.from(JSON.stringify(response)), {
      correlationId: msg.properties.correlationId,
    });
    console.log("Response sent:", sent);

    channel.ack(msg);
  });
}
```

4) Видим, что сообщения отправляются и ловятся между сервисами

```
Auth service started
RabbitMQ connected: auth
Server is running at http://localhost:3000
Received token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZC
I6MywiaWF0IjoxNzI2Njk0MDQyLCJleHAiOjE3MjY3ODA0NDJ9.KcTa9FZQ
pFnzXhhWR8r4B26sUP7k2b7-SkpiBTcTpN8
Response sent: true
```

```
[nodemon] restarting due to changes...
[nodemon] starting `tsx src/index.ts`
Server started
RabbitMQ connected: auth
Server is running at http://localhost:3001
Response {"valid":true,"user":{"id":3,"iat":1726694042,"exp
":1726780442}}
```

5) Напишем Dockerfile к обоим сервисам

```
FROM node:21-alpine

WORKDIR /app

COPY ./package.json ./

ARG NODE_ENV=production
ENV NODE_ENV=${NODE_ENV}

COPY . .
RUN npm install
RUN npm run build
CMD ["npm", "start"]
```

6) и docker compose

```yaml
version: "3.8"

services:
  # RabbitMQ service
  rabbitmq:
    image: rabbitmq:3.13-management
    container_name: rabbitmq
    hostname: rabbitmq
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      RABBITMQ_DEFAULT_USER: guest
      RABBITMQ_DEFAULT_PASS: guest

  # Auth service
  auth:
    build:
      context: ./auth
      dockerfile: Dockerfile
    container_name: auth
    restart: always
    ports:
      - "3000:3000"
    depends_on:
      - rabbitmq
    environment:
      - PG_HOST=host.docker.internal
```

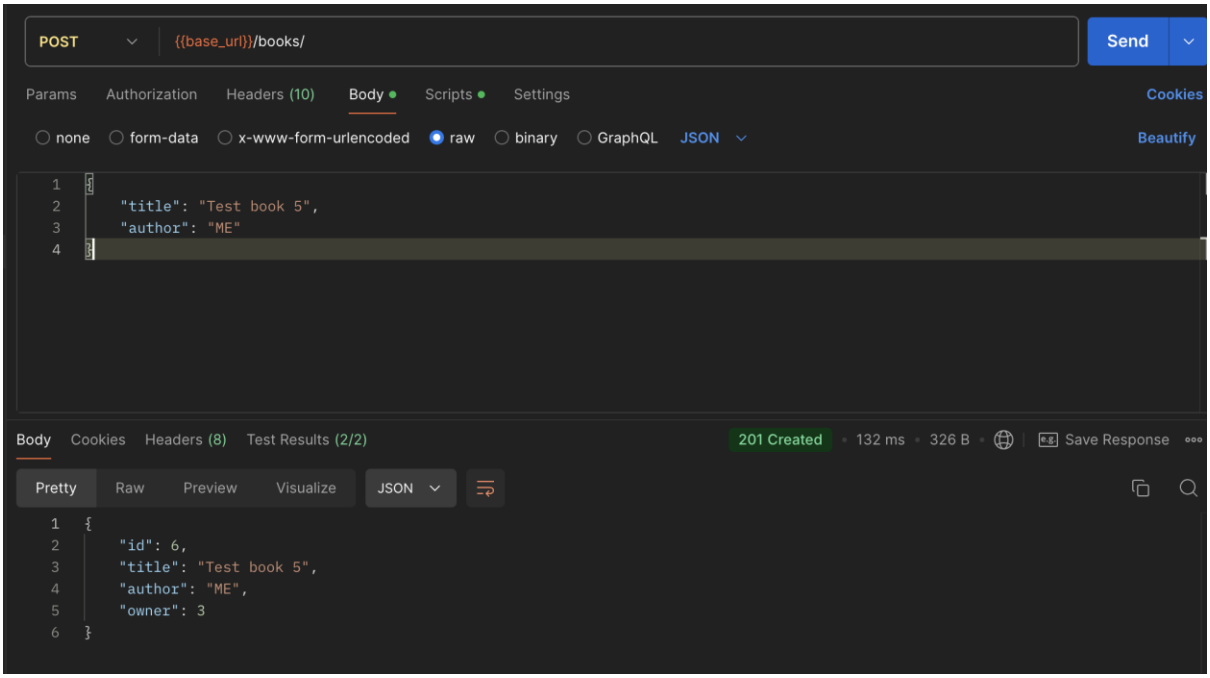
```
      - PG_PORT=5432
      - PG_USER=postgres
      - PG_DB=itmo_backend
      - PG_PASSWORD=postgres
      - JWT_SECRET=123456
      - AMQP=amqp://rabbitmq:5672

  # Server service
  server:
    build:
      context: ./server
      dockerfile: Dockerfile
    container_name: server
    restart: always
    ports:
      - "3001:3001"
    depends_on:
      - rabbitmq
    environment:
      - PG_HOST=host.docker.internal
      - PG_PORT=5432
      - PG_USER=postgres
      - PG_DB=itmo_backend
      - PG_PASSWORD=postgres
      - JWT_SECRET=123456
```

7) Все контейнеры успешно подняты

Name	Image	Status	Port(s)	Last started	CPU (%)	Actions
backend_itmo		Running (3/3)		20 seconds ago	3.44%	
rabbitmq b67148ed08f8	rabbitmq:3.13-management	Running	15672:15672 Show all ports (2	20 seconds ago	1.71%	
auth 858bdd2fe485	backend_itmo-auth	Running	3000:3000	20 seconds ago	1.73%	
server a51f9e4b5cf4	backend_itmo-server	Running	3001:3001	20 seconds ago	0%	

8) Проверяем что все работает



Вывод

В ходе данной работы была реализована связь микросервисов с помощью брокера сообщений RabbitMQ, а так же все сервисы были собраны и запущены в докер контейнерах