САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет по лабораторной работе “Docker, очереди сообщений”

Выполнил:
Пронина Мария
Соколовская Арина

Группа:
К33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задание:

Необходимо упаковать приложение в docker-контейнеры и обеспечить сетевое взаимодействие между различными частями приложения, а также настроить общение микросервисов между собой посредством RabbitMQ.

Ход работы:

1. Создание сервисов для работы с RabbitMQ:

Сервис для producer микросервиса hackathon_app:

```
import amqp, { Connection, Channel } from 'amqplib';


class RabbitMQService {
    connection!: Connection;
    channel!: Channel;

    async connect() {
        if (this.connection && this.channel) return;

        try {
            this.connection = await amqp.connect('amqp://localhost');
            this.channel = await this.connection.createChannel();
        } catch (error) {
            console.log(`Error connecting to RabbitMQ: ${error}`);
        }
    }

    async sendMessage(queue: string, message: any) {
```

```
        try {
            if (!this.channel) {
                await this.connect();
            }

            this.channel.sendToQueue(queue,
Buffer.from(JSON.stringify(message)));
        } catch (error) {
            console.log(`Error sending message to RabbitMQ: ${error}`);
        }
    }
}

const rabbitMQService = new RabbitMQService();

export default rabbitMQService;
```

Сервис для работы consumer микросервиса auth:

```
import amqp, { Connection, Channel } from 'amqplib';

class RabbitMQService {
    private connection: Connection | null = null;
    private channel: Channel | null = null;

    async connect(): Promise<void> {
```

```
        if (this.connection && this.channel) return;


        try {
            this.connection = await amqp.connect('amqp://localhost');
            this.channel = await this.connection.createChannel();
        } catch (err) {
            console.log(err);
        }
    }


    async consumeMessage(queue: string, callback: (msg:
amqp.ConsumeMessage | null) => void): Promise<void> {
        if (!this.channel) {
            throw new Error('Channel is not initialized');
        }


        await this.channel.consume(queue, callback, { noAck: true });
    }


    async createQueue(queue: string): Promise<void> {
        if (!this.channel) {
            throw new Error('Channel is not initialized');
        }


        await this.channel.assertQueue(queue);
```

```
    }

}



const mqConnection = new RabbitMQService();

export default mqConnection;
```

Consumer микросервис начинает принимать сообщения в очереди при запуске приложения. Producer отсылает в очередь сообщение с id пользователя при создании новой команды или добавлении пользователя в команду.

```
version: '3.8'

services:
 auth:
   build:
     dockerfile: auth.Dockerfile
   ports:
     - "8081:8081"
   environment:
     - NODE_ENV=production
     - SERVICE_PORT=8081
   depends_on:
     - hackathon
     - rabbitmq


 hackathon:
   build:
     dockerfile: app.Dockerfile
   ports:
     - "8080:8080"
   environment:
     - NODE_ENV=production
     - SERVICE_PORT=8080
   depends_on:
     - rabbitmq

 gateway:
   build:
     dockerfile: gateway.Dockerfile
   ports:
     - "8000:8000"
   environment:
     - NODE_ENV=production
     - SERVICE_PORT=8000
   depends_on:
     - hackathon
     - auth
```

```
  rabbitmq:
     image: rabbitmq
     ports:
         - "5672:5672"
     healthcheck:
       test: rabbitmq-diagnostics -q ping
       interval: 5s
       timeout: 5s
       retries: 5
```

С помощью docker compose файла мы поднимаем все образы, которые нужны для функционирования лабораторной работы.

```
FROM node:18-alpine

WORKDIR /auth

COPY ./auth/package.json .

RUN npm install

COPY ./auth .

RUN npm run build

CMD ["npm", "start"]
```

Это пример docker файла для создания образа микросервера auth. Остальные образы собираются аналогичным образом.

Вывод:

В ходе лабораторной работы микросервисы были подняты в docker. Также было реализовано общение микросервисов с использованием очереди сообщений RabbitMQ.