

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа 5

Выполнил:

Котовщиков Андрей

К3339

Проверил:

Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

Реализовать межсерверное взаимодействие посредством очередей сообщений.

Ход работы

1. Подключение и настройка Kafka

В качестве брокера сообщений был выбран Apache Kafka. Для развертывания всей необходимой инфраструктуры был использован docker-compose (рисунок 1).

```
1  services:
2    > Run Service
3    zookeeper:
4      image: confluentinc/cp-zookeeper:7.4.0
5      container_name: zookeeper
6      ports:
7        - "2181:2181"
8      environment:
9        ZOOKEEPER_CLIENT_PORT: 2181
10     networks:
11       - mail_net
12
13     > Run Service
14     kafka:
15       image: confluentinc/cp-kafka:7.4.0
16       container_name: kafka
17       healthcheck:
18         test: ["CMD", "bash", "-c", "echo > /dev/tcp/localhost/9092"]
19         interval: 5s
20         timeout: 5s
21         retries: 5
22       depends_on:
23         - zookeeper
24       ports:
25         - "9092:9092"
26       environment:
27         KAFKA_BROKER_ID: 1
28         KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
29         KAFKA_LISTENERS: "PLAINTEXT://0.0.0.0:9092"
30         KAFKA_ADVERTISED_LISTENERS: "PLAINTEXT://kafka:9092"
31         KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
32       networks:
33         - mail_net
```

Рисунок 1 – Настройка kafka в docker-compose

2. Реализация межсервисного взаимодействия

2.1. Код потребителя (consumer)

В качестве потребителя (consumer) выступает микросервис для рассылки email писем, написанный на nodejs. Код представлен на рисунке 2. Сервис слушает kafka топик и читает сообщения, закодированные в JSON формат. Происходит десериализация данных, из которых достается адрес получателя и код, который необходимо отправить на почту. Далее через smtp протокол происходит отправка письма с кодом.

```
21 const kafka = new Kafka({
22   clientId: 'mail-sender',
23   brokers: KAFKA_BROKERS.split(','),
24 });
25
26 const consumer = kafka.consumer({ groupId: 'mail-group' });
27
28 const transporter = nodemailer.createTransport({
29   host: MAIL_HOST,
30   port: Number(MAIL_PORT),
31   secure: false,
32   requireTLS: true,
33   auth: {
34     user: MAIL_USER,
35     pass: MAIL_PASS,
36   },
37   tls: {
38     minVersion: 'TLSv1.2',
39   },
40 });
41
42 async function sendMail(to: string, code: string) {
43   const info = await transporter.sendMail({
44     from: `"${MAIL_FROM_NAME}" <${MAIL_USER}>`,
45     to,
46     subject: 'Ваш одноразовый код',
47     text: `Ваш код: ${code}`,
48     html: `<p>Ваш код: <b>${code}</b></p>`,
49   });
50   console.log(`Message sent: ${info.messageId}`);
51 }
52
53 async function run() {
54   await consumer.connect();
55   // @ts-ignore
56   await consumer.subscribe({ topic: KAFKA_TOPIC, fromBeginning: false });
57   console.log(`Subscribed to topic ${KAFKA_TOPIC}`);
58
59   await consumer.run({
60     eachMessage: async ({ message }) => {
61       try {
62         if (!message.value) return;
63         const payload = JSON.parse(message.value.toString());
64         const { email, code } = payload;
65         console.log(`Received: ${email}, code=${code}`);
66         await sendMail(email, code);
67       } catch (err) {
68         console.error('Error processing message', err);
69       }
70     },
71   });
72 }
```

Рисунок 2 – Чтение сообщений из kafka

2.2. Код отправителя (producer)

В качестве отправителя данных (producer) выступает сервис с фильмами. При регистрации нового пользователя в системе генерируется специальный код. Затем в формате json кодируется сообщение с указанным при регистрации email адресом и сгенерированным кодом. В конце сообщение отправляется в топик kafka. Код отправки сообщения в брокер представлен на рисунке 3 и 4.

```
48 class ExternalMailSender(MailSender):
49     def __init__(self, broker: BrokerClient) -> None:
50         self._broker = broker
51
52     async def send_code(self, code: str, email: str) -> None:
53         broker_message = {
54             "email": email,
55             "code": code,
56         }
57
58         await self._broker.publish(
59             queue_name=config.MAILER_QUEUE_NAME,
60             message=json.dumps(broker_message),
61         )
```

Рисунок 3 – Логика отправки письма через брокер

```
34 class KafkaClient(BrokerClient):
35     def __init__(self, broker_url: str) -> None:
36         self._broker_url = broker_url
37         self._producer: Optional[AIOKafkaProducer] = None
38
39     async def connect(self) -> None:
40         self._producer = AIOKafkaProducer(bootstrap_servers=self._broker_url)
41         await self._producer.start()
42
43     async def publish(self, queue_name: str, message: str) -> None:
44         assert self._producer is not None, "Call connect before publish"
45         await self._producer.send_and_wait(topic=queue_name, value=message.encode())
46
47     async def close(self) -> None:
48         if self._producer is not None:
49             await self._producer.stop()
```

Рисунок 4 – Логика адаптера для взаимодействия с kafka

Вывод

В ходе выполнения домашней работы номер 5 была выстроена асинхронная коммуникация между микросервисами с помощью брокера сообщений kafka.