САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №5

Выполнил:

Пиотуховский Александр

K3441

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

Задача

- подключить и настроить kafka;
- реализовать межсервисное взаимодействие посредством kafka.

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).

```yaml
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.4.0
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
    networks:
      - mail_net

  kafka:
    image: confluentinc/cp-kafka:7.4.0
    container_name: kafka
    healthcheck:
      test: ["CMD", "bash", "-c", "echo > /dev/tcp/localhost/9092"]
      interval: 5s
      timeout: 5s
      retries: 5
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
      KAFKA_LISTENERS: "PLAINTEXT://0.0.0.0:9092"
      KAFKA_ADVERTISED_LISTENERS: "PLAINTEXT://kafka:9092"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    networks:
      - 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 деспирализация данных, из которых достается адрес получателя и код, который необходимо отправить на почту. Далее через smtp протокол происходит отправка письма с кодом.

```
const kafka = new Kafka({
    clientId: 'mail-sender',
    brokers: KAFKA_BROKERS.split(','),
});

const consumer = kafka.consumer({ groupId: 'mail-group' });

const transporter = nodemailer.createTransport({
    host: MAIL_HOST,
    port: Number(MAIL_PORT),
    secure: false,
    requireTLS: true,
    auth: {
        user: MAIL_USER,
        pass: MAIL_PASS,
    },
    tls: {
        minVersion: 'TLSv1.2',
    },
});

async function sendMail(to: string, code: string) {
    const info = await transporter.sendMail({
        from: `"${MAIL_FROM_NAME}" <${MAIL_USER}>`,
        to,
        subject: 'Ваш одноразовый код',
        text: `Ваш код: ${code}`,
        html: `<p>Ваш код: <b>${code}</b></p>`,
    });
    console.log(`Message sent: ${info.messageId}`);
}

async function run() {
    await consumer.connect();
    // @ts-ignore
    await consumer.subscribe({ topic: KAFKA_TOPIC, fromBeginning: false });
    console.log(`Subscribed to topic ${KAFKA_TOPIC}`);

    await consumer.run({
        eachMessage: async ({ message }) => {
            try {
                if (!message.value) return;
                const payload = JSON.parse(message.value.toString());
                const { email, code } = payload;
                console.log(`Received: ${email}, code=${code}`);
                await sendMail(email, code);
            } catch (err) {
```

Рисунок 2 – Чтение сообщений из kafka

2.2. Код отправителя (producer)

В качестве отправителя данных (producer) выступает сервис с фильмами. При определённых ситуациях в системе генерируется специальных код. Затем в формате Json кодируется сообщение с указанным

email адресом. В конце сообщение отправляется в топик kafka. Код отправки сообщения в брокер представлен на рисунке 3 и 4.

```python
class ExternalMailSender(MailSender):  1 usage
    def __init__(self, broker: BrokerClient) -> None:
        self._broker = broker

    async def send_code(self, code: str, email: str) -> None:
        broker_message = {
            "email": email,
            "code": code,
        }

        await self._broker.publish(
            queue_name=config.MAILER_QUEUE_NAME,
            message=json.dumps(broker_message),
        )
```

Рисунок 3 – Логика отправки письма через брокер

```python
class KafkaClient(BrokerClient):  1 usage
    def __init__(self, broker_url: str) -> None:
        self._broker_url = broker_url
        self._producer: Optional[AIOKafkaProducer] = None

    async def connect(self) -> None:
        self._producer = AIOKafkaProducer(bootstrap_servers=self._broker_url)
        await self._producer.start()

    async def publish(self, queue_name: str, message: str) -> None:
        assert self._producer is not None, "Call connect before publish"
        await self._producer.send_and_wait(topic=queue_name, value=message.encode())

    async def close(self) -> None:
        if self._producer is not None:
            await self._producer.stop()
```

Рисунок 4 – Логика адаптера для взаимодействия с kafka.

Вывод

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