

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа 5

Реализация межсервисного взаимодействия посредством
очередей сообщений

Выполнила:
Едигарева Дарья

Группа
К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

подключить и настроить rabbitMQ/kafka;

реализовать межсервисное взаимодействие посредством rabbitMQ/kafka.

Ход работы

В docker-compose добавлен брокер и healthcheck, чтобы сервисы стартовали только после готовности RabbitMQ.

```
rabbitmq:

  image: rabbitmq:3.13-management

  container_name: message_broker

  environment:

    RABBITMQ_DEFAULT_USER: guest

    RABBITMQ_DEFAULT_PASS: guest

  ports:

    - '5672:5672'

    - '15672:15672'

  volumes:

    - rabbit-data:/var/lib/rabbitmq

  healthcheck:

    test: ["CMD", "rabbitmq-diagnostics", "status"]

    interval: 10s

    timeout: 10s

    retries: 6
```

Все сервисы зависят от rabbitmq:

```
auth-service:

  build:

  ...

  depends_on:

    rabbitmq:

      condition: service_healthy
```

Во всех сервисах используется один и тот же помощник:

```
"use strict";

var __importDefault = (this && this.__importDefault) || function (mod) {
    return (mod && mod.__esModule) ? mod : { "default": mod };
};

Object.defineProperty(exports, "__esModule", { value: true });

exports.MessageBroker = void 0;

const amqplib_1 = __importDefault(require("amqplib"));

const settings_1 = __importDefault(require("../config/settings"));

const SERVICE_NAME = '[auth-service]';

class MessageBroker {

    async getConnection() {

        if (this.connection) {

            return this.connection;

        }

        const connection = await
amqplib_1.default.connect(settings_1.default.MESSAGE_BROKER_URL);

        connection.on('error', (err) => {

            console.error(`${SERVICE_NAME} RabbitMQ connection error`, err);
```

```
});

connection.on('close', () => {

    console.warn(`${SERVICE_NAME} RabbitMQ connection closed`);

    this.connection = undefined;

    this.channel = undefined;

});

this.connection = connection;

return connection;

}

async getChannel() {

    if (this.channel) {

        return this.channel;

    }

    const connection = await this.getConnection();

    const channel = await connection.createChannel();

    await channel.prefetch(10);

    this.channel = channel;

    return channel;

}

async connect() {

    await this.getChannel();

    console.log(`${SERVICE_NAME} RabbitMQ connected`);

}

async publish(queue, message) {

    const channel = await this.getChannel();

    await channel.assertQueue(queue, { durable: true });
```

```

        channel.sendToQueue(queue, Buffer.from(JSON.stringify(message)), {
persistent: true });

    }

    async consume(queue, handler) {

        const channel = await this.getChannel();

        await channel.assertQueue(queue, { durable: true });

        await channel.consume(queue, async (msg) => {

            if (!msg) {

                return;

            }

            try {

                const payload = JSON.parse(msg.content.toString());

                await handler(payload);

                channel.ack(msg);

            }

            catch (err) {

                console.error(`${SERVICE_NAME} Failed to process ${queue} message`,
err);

                channel.nack(msg, false, false);

            }

        });

    }

}

exports.MessageBroker = MessageBroker;

```

MessageBroker берёт адрес RabbitMQ из настроек, лениво открывает одно соединение и канал, следит за ошибками/закрытием (сбрасывает ссылки),

логирует успешное подключение, а дальше делает publish, гарантируя очередь через assertQueue(...durable) и отправляет JSON с persistent=true, и consume, где для каждой очереди создаётся обработчик. Обработчик парсит JSON и дает ack при успехе либо nack без повторной постановки при ошибке. сервисы обмениваются событиями, не теряя сообщения и не плодя лишних подключений.

Сценарии межсервисного взаимодействия

1. Регистрация - авто-создание профиля

Публикация:

```
try {

    await broker.publish(QUEUES.USER_REGISTERED, { id: saved.id, email:
saved.email });

    console.log(`[auth-service] Published user.registered for ${saved.email}`);

} catch (err) {

    console.error('[auth-service] Failed to publish user.registered event', err);

}
```

Код консьюмера:

```
await broker.connect();

await broker.consume<UserRegisteredPayload>(QUEUES.USER_REGISTERED, async
(payload) => {

    if (!payload?.id) {

        console.warn('[profiles-service] user.registered payload without id');

        return;

    }

    try {

        const profileService = container.resolve<IProfileService>(PROFILE_SERVICE);

        const existing = await profileService.getMy(payload.id);

        if (existing) {
```

```
        return;

    }

    const fallbackName = payload.email ?? 'New user';

    await profileService.createForUser(payload.id, { fullName: fallbackName });

    console.log(`[profiles-service] Auto-created profile for ${payload.id}`);

    } catch (err) {

        console.error('[profiles-service] Failed to create profile from user.registered', err);

    }

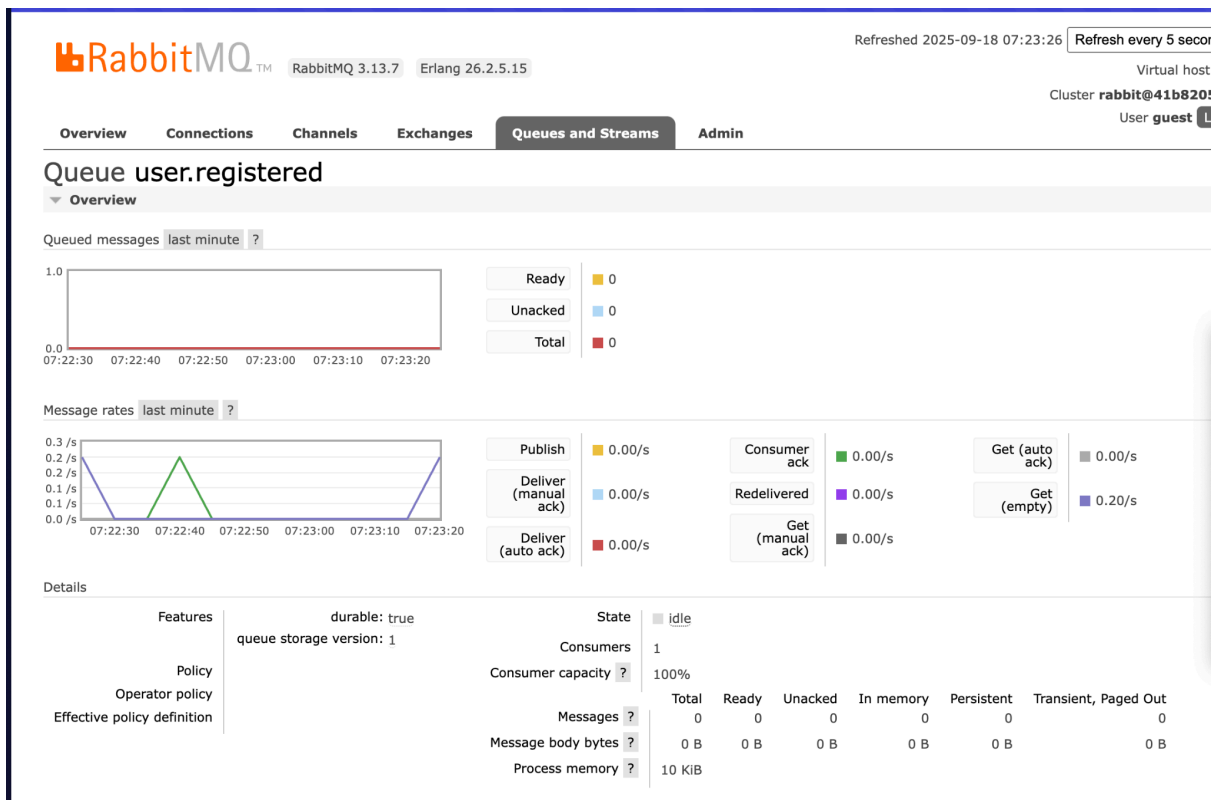
});
```

Бизнес-ценность: регистрация не блокируется ожиданием второго сервиса, профиль появится автоматически даже если profiles запускается позже.

```
[profiles-service] RabbitMQ connected
```

```
[auth-service] Published user.registered for 79fa8c5a-8ebb-4af7-91ff-0848bc040215
```

```
[profiles-service] Auto-created profile for 79fa8c5a-8ebb-4af7-91ff-0848bc040215
```



2. Изменение резюме - уведомление вакансий

Публикация:

```
try {  
  
    await broker.publish(QUEUES.RESUME_UPDATED, { userId, resumeId, action,  
...meta });  
  
} catch (err) {  
  
    console.error('[profiles-service] Failed to publish resume.updated event',  
err);  
  
}
```

Аналогично при update/delete/addExperience/addEducation меняется action.

Обработка:

```
await broker.connect();  
  
await broker.consume<ResumeUpdatedPayload>(QUEUES.RESUME_UPDATED, async (payload)  
=> {  
  
    if (!payload?.userId || !payload?.resumeId) {
```



```
    console.warn('[vacancies-service] resume.updated payload missing userId or  
resumeId');  
  
    return;  
  
  }  
  
  console.log('[vacancies-service] resume.updated received', payload);  
  
});
```

Бизнес-ценность: сервис вакансий оперативно узнаёт об активности соискателей и может реагировать (например, отправлять рекомендации работодателям) без опроса базы

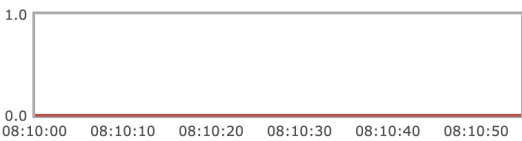
```
[vacancies-service] Published employer.profile.created for 9554655e-8025-44c2-8036-f5094ddfd94c
[vacancies-service] resume.updated received {
  userId: '9554655e-8025-44c2-8036-f5094ddfd94c',
  resumeId: '6f6219f7-c9e8-4fd6-9f80-2bfb0868839e',
  action: 'created'
}
[vacancies-service] resume.updated received {
  userId: '9554655e-8025-44c2-8036-f5094ddfd94c',
  resumeId: '0d4c637d-89a9-4154-9def-692cc1de4e50',
  action: 'created'
}
[vacancies-service] resume.updated received {
  userId: '9554655e-8025-44c2-8036-f5094ddfd94c',
  resumeId: '0d4c637d-89a9-4154-9def-692cc1de4e50',
  action: 'updated'
}
[vacancies-service] resume.updated received {
  userId: '9554655e-8025-44c2-8036-f5094ddfd94c',
  resumeId: '0d4c637d-89a9-4154-9def-692cc1de4e50',
  action: 'experience_added',
  experienceId: '52134397-7572-4483-9805-94cb8e9d224e'
}
[vacancies-service] resume.updated received {
  userId: '9554655e-8025-44c2-8036-f5094ddfd94c',
  resumeId: '0d4c637d-89a9-4154-9def-692cc1de4e50',
  action: 'education_added',
  educationId: 'c611a551-49eb-4a49-b44e-bbd9ac28916f'
}
```

profile

Queue resume.updated

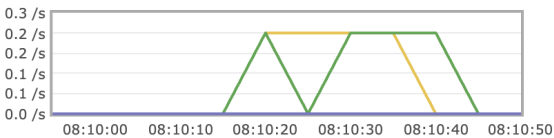
Overview

Queued messages last minute ?



Ready	0
Unacked	0
Total	0

Message rates last minute ?



Publish	0.00/s	Consumer ack	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s

Details

Features	State	idle
Policy	Consumers	1
Operator policy	Consumer capacity ?	100%
Effective policy definition	Messages ?	0
	Message body bytes ?	0 B
	Process memory ?	10 KiB

Consumers (1)

Bindings (1)

3. Создание профиля работодателя - обновление роли

Публикация:

```
try {

    await broker.publish(QUEUES.EMPLOYER_PROFILE_CREATED, {

        userId,

        companyId: company.id,

        phone: dto.phone,

    });

    console.log(`[vacancies-service] Published employer.profile.created for ${userId}`);

} catch (err) {

    console.error('[vacancies-service] Failed to publish employer.profile.created event', err);

}
```

Обработка:

```
await broker.connect();

await broker.consume<ResumeUpdatedPayload>(QUEUES.RESUME_UPDATED, async (payload) => {

    if (!payload?.userId || !payload?.resumeId) {

        console.warn('[vacancies-service] resume.updated payload missing userId or resumeId');

        return;

    }

    console.log('[vacancies-service] resume.updated received', payload);

});
```

Бизнес-ценность: сервис авторизации автоматически синхронизирует роль, пользователю не нужно перелогиниваться.

```
[vacancies-service] Published employer.profile.created for 9554655e-8025-44c2-8036-f5094ddfd94c
```


```
[auth-service] Role updated to employer for user 9554655e-8025-44c2-8036-f5094ddfd94c
```

Queue employer.profile.created

Overview

Queued messages

last minute ?



Ready

Unacked

Total

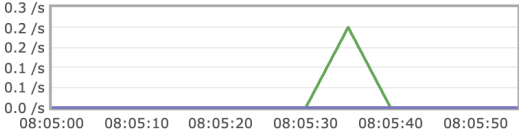
0

0

0

Message rates

last minute ?



Publish

Deliver (manual ack)

Deliver (auto ack)

0.00/s

0.00/s

0.00/s

Consumer ack

Redelivered

Get (manual ack)

0.00/s

0.00/s

0.00/s

Details

Features

Policy

Operator policy

Effective policy definition

State

Consumers

Consumer capacity ?

Messages ?

Message body bytes ?

Process memory ?

idle

1

100%

Total

Ready

Unacked

In memory

0

0

0

0

0 B

0 B

0 B

0 B

10 KiB

Consumers (1)

Bindings (1)

Вывод:

В проект добавлен RabbitMQ. Auth, Profiles и Vacancies обмениваются событиями user.registered, resume.updated, employer.profile.created. Профили создаются автоматически, роль работодателя обновляется без ручных вызовов, активность соискателей приходит в сервис вакансий.