САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №5
Очереди сообщений

Выполнил:
Даньшин Семён
К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Подключить и настроить RabbitMQ/Kafka. Реализовать межсервисное взаимодействие посредством RabbitMQ/Kafka.

Ход работы

1. Выбор технологии

Для реализации очередей сообщений был выбран **Apache Kafka** как более масштабируемое и производительное решение для микросервисной архитектуры.

2. Настройка Kafka

Docker Compose конфигурация (kafka/kafka-compose.yaml):

```
services:
  kafka-ui:
    container_name: fitness-kafka-ui
    profiles: [kafka, full, dev]
    image: provectuslabs/kafka-ui:latest
    ports:
      - "8090:8080"
    environment:
      KAFKA_CLUSTERS_0_NAME: local
      KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: kafka0:29092

  kafka0:
    container_name: fitness-kafka
    profiles: [kafka, full, dev]
    image: confluentinc/cp-kafka:7.7.1.arm64
    ports:
      - "9092:9092"
    environment:
      KAFKA_NODE_ID: 1
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://kafka0:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_LISTENERS:
PLAINTEXT://kafka0:29092,CONTROLLER://kafka0:29093,PLAINTEXT_HOST://:9092
      KAFKA_CONTROLLER_LISTENER_NAMES: "CONTROLLER"
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_PROCESS_ROLES: "broker,controller"
      KAFKA_LOG_DIRS: "/tmp/kraft-combined-logs"
      CLUSTER_ID: 'MkU3OEVBNTcwNTJENDM2Qk'
```

3. Реализация Producer в Go Backend

Kafka Producer (backend/internal/clients/kafka/producer.go):

```
type Producer interface {
    Publish(ctx context.Context, topic, key string, message interface{})
error
}
```

```
type kafkaProducer struct {
    writer *kafka.Writer
}

func NewProducer(brokers []string, topic string) Producer {
    return &kafkaProducer{
        writer: &kafka.Writer{
            Addr:     kafka.TCP(brokers...),
            Topic:    topic,
            Balancer: &kafka.LeastBytes{},
        },
    }
}

func (p *kafkaProducer) Publish(ctx context.Context, topic, key string,
message interface{}) error {
    span, ctx := opentracing.StartSpanFromContext(ctx, "kafka.Publish")
    defer span.Finish()

    messageBytes, err := json.Marshal(message)
    if err != nil {
        return err
    }

    return p.writer.WriteMessages(ctx, kafka.Message{
        Topic: topic,
        Key:   []byte(key),
        Value: messageBytes,
    })
}
```

Email Service Integration:

```
func (s *Service) SendWelcomeEmail(ctx context.Context, email, name string)
error {
    span, ctx := opentracing.StartSpanFromContext(ctx,
"service.email.SendWelcomeEmail")
    defer span.Finish()

    payload := domain.WelcomePayload{
        Email: email,
        Name:  name,
    }
    message := domain.EmailMessage{
        Type:    domain.WelcomeEmail,
        Payload: payload,
    }
    return s.producer.Publish(ctx, s.topic, email, message)
}
```

4. Реализация Consumer в Email сервисе

Email Service Consumer (email/src/consumers/kafkaConsumer.ts):

```
export async function runConsumer(): Promise<void> {
  const consumer = kafka.consumer({
    groupId: config.kafka.clientId,
    sessionTimeout: 30000,
    heartbeatInterval: 3000
```

```
  });

  await consumer.connect();
  await consumer.subscribe({ topic: config.kafka.topic });

  await consumer.run({
    eachMessage: async ({ topic, partition, message }) => {
      const tracer = trace.getTracer('email-consumer');
      const span = tracer.startSpan('process_message');

      try {
        const messageValue = message.value?.toString();
        if (!messageValue) {
          throw new Error('Empty message received');
        }

        const emailMessage: EmailMessage = JSON.parse(messageValue);

        span.setAttributes({
          'message.type': emailMessage.type,
          'message.topic': topic,
          'message.partition': partition
        });

        await handleEmailMessage(emailMessage);

        span.setStatus({ code: SpanStatusCode.OK });
      } catch (error: any) {
        span.recordException(error);
        span.setStatus({
          code: SpanStatusCode.ERROR,
          message: error.message
        });
        throw error;
      } finally {
        span.end();
      }
    },
  });
}
```

5. Типы сообщений и обработчики

Типы email сообщений:

```
export enum EmailType {
  WELCOME = 'welcome',
  PASSWORD_RESET = 'password_reset',
  WORKOUT_REMINDER = 'workout_reminder',
  ACHIEVEMENT = 'achievement'
}

export interface EmailMessage {
  type: EmailType;
  payload: WelcomePayload | PasswordResetPayload | WorkoutReminderPayload;
}

export interface WelcomePayload {
  email: string;
  name: string;
```

```
}
```

Обработчики сообщений:

```
async function handleEmailMessage(message: EmailMessage): Promise<void> {
  logger.info('Processing email message', { type: message.type });

  switch (message.type) {
    case EmailType.WELCOME:
      await handleWelcomeEmail(message.payload as WelcomePayload);
      break;
    case EmailType.PASSWORD_RESET:
      await handlePasswordResetEmail(message.payload as
PasswordResetPayload);
      break;
    case EmailType.WORKOUT_REMINDER:
      await handleWorkoutReminderEmail(message.payload as
WorkoutReminderPayload);
      break;
    default:
      logger.warn('Unknown email type', { type: message.type });
  }
}

async function handleWelcomeEmail(payload: WelcomePayload): Promise<void> {
  await sendMail({
    to: payload.email,
    subject: 'Welcome to Fitness Trainer!',
    templateName: 'welcome',
    context: {
      name: payload.name,
      appName: 'Fitness Trainer'
    }
  });
}
```

6. Шаблоны email сообщений

Welcome Email Template (email/src/templates/welcome.hbs):

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome to {{appName}}</title>
</head>
<body>
    <h1>Welcome, {{name}}!</h1>
    <p>Thank you for joining {{appName}}. We're excited to help you on your
fitness journey!</p>

    <h2>Getting Started:</h2>
    <ul>
        <li>Create your first workout routine</li>
        <li>Log your exercises and track progress</li>
        <li>Set your fitness goals</li>
    </ul>

    <p>Happy training!</p>
</body>
</html>
```

7. Конфигурация и переменные окружения

Email Service Configuration (.env):

```
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_SECURE=false
SMTP_USER=your_email@gmail.com
SMTP_PASS=your_app_password
EMAIL_FROM=noreply@fitnesstrainer.com

KAFKA_BROKERS=localhost:9092
KAFKA_CLIENT_ID=email-service
KAFKA_TOPIC=email-topic
```

8. Мониторинг и трассировка

OpenTelemetry трассировка:

```
// Tracing для Kafka consumer
export async function sendMail({ to, subject, templateName, context }:
SendMailOptions): Promise<void> {
  const tracer = trace.getTracer('email-service');

  const renderSpan = tracer.startSpan('render_email_template');
  renderSpan.setAttribute('template.name', templateName);

  try {
    const html = renderTemplate(templateName, context);
    renderSpan.setStatus({ code: SpanStatusCode.OK });
  } catch (error: any) {
    renderSpan.recordException(error);
    renderSpan.setStatus({
      code: SpanStatusCode.ERROR,
      message: error.message
    });
    throw error;
  } finally {
    renderSpan.end();
  }

  const sendSpan = tracer.startSpan('send_email');
  sendSpan.setAttribute('email.to', to);
  sendSpan.setAttribute('email.subject', subject);

  try {
    await transporter.sendMail({
      from: emailConfig.from,
      to,
      subject,
      html
    });
    sendSpan.setStatus({ code: SpanStatusCode.OK });
  } catch (error: any) {
    sendSpan.recordException(error);
    sendSpan.setStatus({
      code: SpanStatusCode.ERROR,
      message: error.message
    });
```

```
    throw error;
  } finally {
    sendSpan.end();
  }
}
```

9. Обработка ошибок и retry логика

Retry механизм для Kafka:

```
const retryOptions = {
  retries: 3,
  retryDelayInMs: 1000,
  factor: 2,
  maxRetryTime: 30000
};

await consumer.run({
  eachMessage: async ({ topic, partition, message }) => {
    let attempts = 0;
    const maxAttempts = retryOptions.retries + 1;

    while (attempts < maxAttempts) {
      try {
        await handleEmailMessage(emailMessage);
        break;
      } catch (error) {
        attempts++;
        if (attempts >= maxAttempts) {
          logger.error('Max retry attempts reached', { error, attempts });
          throw error;
        }

        const delay = retryOptions.retryDelayInMs *
Math.pow(retryOptions.factor, attempts - 1);
        await new Promise(resolve => setTimeout(resolve, delay));
      }
    }
  }
});
```

10. Docker интеграция

Email Service Dockerfile:

```
FROM node:slim AS builder
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY tsconfig.json ./
COPY src ./src
RUN npm run build

FROM node:slim AS production
WORKDIR /app
COPY package*.json ./
RUN npm install --omit=dev
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/src/templates ./dist/templates
```

```
ENV NODE_ENV=production
CMD ["node", "dist/index.js"]
```

Композитная настройка (compose.yaml):

```
email:
  container_name: email
  profiles: [backend, full]
  build:
    context: email
    dockerfile: Dockerfile
  env_file:
    - ./email/.env.docker
  expose:
    - "8081"
  depends_on:
    - kafka0
```

[Скриншот Kafka UI с топиками и сообщениями]

[Скриншот логов email сервиса]

11. Тестирование межсервисного взаимодействия

Интеграционный тест:

```
describe('Email Service Integration', () => {
  it('should send welcome email on user registration', async () => {
    // 1. Регистрируем пользователя через API
    const user = await registerUser({
      email: 'test@example.com',
      name: 'Test User'
    });

    // 2. Проверяем, что сообщение попало в Kafka
    await waitForKafkaMessage('email-topic');

    // 3. Проверяем, что email был отправлен
    await waitForEmailDelivery('test@example.com');
  });
});
```

Вывод

Успешно реализовано межсервисное взаимодействие с использованием Apache Kafka:

1. **Настроена Kafka** с UI для мониторинга
2. **Реализован Producer** в Go backend для отправки сообщений
3. **Создан Consumer** в TypeScript email сервисе
4. **Определены типы сообщений** и их обработчики
5. **Добавлена трассировка** с OpenTelemetry
6. **Реализован retry механизм** для обработки ошибок
7. **Интегрировано в Docker Compose** для легкого развертывания
8. **Покрыто тестами** для проверки корректности работы

Система обеспечивает надежную асинхронную коммуникацию между сервисами и может быть легко расширена для новых типов сообщений.