САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №3
Микросервисы

Выполнил:
Даньшин Семён
К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Выделить самостоятельные модули в приложении. Провести разделение API на микросервисы (минимум 3). Настроить сетевое взаимодействие между микросервисами.

Ход работы

1. Анализ текущей архитектуры

Проект уже реализован как микросервисная архитектура с следующими сервисами:

2. Архитектура микросервисов

Основные микросервисы:

```
├── Backend Service (Go) - основная бизнес-логика
├── Email Service (Node.js) - обработка email сообщений
├── Frontend Service (Next.js) - пользовательский интерфейс
└── Infrastructure Services:
    ├── PostgreSQL - база данных
    ├── Kafka - очереди сообщений
    ├── Nginx - reverse proxy
    └── Jaeger - трассировка
```

3. Backend Service (Основной микросервис)

Ответственности:

- Управление пользователями и аутентификация
- CRUD операции с упражнениями
- Управление планами тренировок (рутинами)
- Логирование и обработка тренировок
- AI-генерация тренировок
- Управление файлами

Конфигурация (compose.yaml):

```
app:
  container_name: app
  profiles: [backend, full]
  build:
    context: backend
    dockerfile: Dockerfile
    target: final
  env_file:
    - ./backend/.env.docker
  expose:
    - "8080"
  depends_on:
    db:
      condition: service_healthy
    jaeger:
      condition: service_started
```

```
    app_init:
      condition: service_completed_successfully
```

API эндпоинты:

```
/v1/auth/*           - аутентификация
/v1/users/*          - управление пользователями
/v1/exercises/*      - управление упражнениями
/v1/routines/*       - планы тренировок
/v1/workouts/*       - тренировки
/v1/muscle_groups/*  - группы мышц
/v1/files/*          - управление файлами
```

4. Email Service (Микросервис уведомлений)

Ответственности:

- Обработка email сообщений
- Отправка welcome писем
- Уведомления о тренировках
- Рендеринг email шаблонов

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"]
```

Структура сервиса:

```
email/
├── src/
│   ├── index.ts           # Точка входа
│   ├── config/            # Конфигурация
│   ├── consumers/         # Kafka consumers
│   ├── handlers/          # Обработчики сообщений
│   ├── services/          # Бизнес-логика
│   ├── templates/         # Email шаблоны
│   ├── types/             # Типы данных
│   └── utils/             # Утилиты
├── package.json
├── tsconfig.json
└── Dockerfile
```

Конфигурация в compose.yaml:

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

5. Frontend Service (Микросервис пользовательского интерфейса)

Ответственности:

- Веб-интерфейс приложения
- Аутентификация пользователей
- Отображение тренировок и упражнений
- Взаимодействие с Backend API

Dockerfile:

```
FROM node:slim AS base
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

FROM node:slim AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM base AS runtime
COPY --from=build /app/.next ./.next
COPY --from=build /app/public ./public

EXPOSE 3000
CMD ["npm", "start"]
```

Конфигурация в compose.yaml:

```
frontend:
  container_name: frontend
  profiles: [frontend, full]
  build:
    context: frontend
    dockerfile: Dockerfile
  expose:
    - "3000"
  depends_on:
    - app
    - dozzle
```

6. Сетевое взаимодействие между микросервисами

6.1 HTTP/REST коммуникация

Frontend → Backend:

```
// API клиент для взаимодействия с backend
export const authApi = new AuthApi({
    baseURL: process.env.NEXT_PUBLIC_API_URL || "/api",
});

// Примеры запросов
await authApi.v1.userServiceGetMe();
await authApi.v1.workoutServiceStartWorkout(request);
await authApi.v1.routineServiceGetRoutines();
```

Nginx как API Gateway:

```
upstream backend {
    server app:8080;
}

upstream frontend {
    server frontend:3000;
}

location /api/ {
    proxy_pass http://backend/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
}

location / {
    proxy_pass http://frontend/;
    proxy_set_header Host $host;
}
```

6.2 Асинхронная коммуникация через Kafka

Backend → Email Service:

```
// Отправка сообщения в Kafka
func (s *Service) SendWelcomeEmail(ctx context.Context, email, name string)
error {
    payload := domain.WelcomePayload{
        Email: email,
        Name:  name,
    }
    message := domain.EmailMessage{
        Type:    domain.WelcomeEmail,
        Payload: payload,
    }
    return s.producer.Publish(ctx, s.topic, email, message)
}
```

Email Service Consumer:

```
// Обработка сообщений из Kafka
await consumer.run({
  eachMessage: async ({ topic, partition, message }) => {
    const emailMessage: EmailMessage =
JSON.parse(message.value?.toString());
    await handleEmailMessage(emailMessage);
  },
});
```

7. Service Discovery и конфигурация

Использование Docker Compose Networks:

```
networks:
  default:
    name: fitness-trainer-network
```

Взаимодействие по именам сервисов:

- `app:8080` - Backend Service
- `email:8081` - Email Service
- `frontend:3000` - Frontend Service
- `kafka0:29092` - Kafka Broker
- `db:5432` - PostgreSQL

8. Мониторинг и трассировка

Jaeger для distributed tracing:

```
jaeger:
  container_name: jaeger
  profiles: [tracing, full, dev]
  image: jaegertracing/all-in-one
  ports:
    - "16686:16686"
    - "4317:4317"
    - "4318:4318"
```

Трассировка в Go:

```
span, ctx := opentracing.StartSpanFromContext(ctx, "service.CreateUser")
defer span.Finish()
```

Трассировка в TypeScript:

```
const tracer = trace.getTracer('email-service');
const span = tracer.startSpan('process_message');
```

9. Health Checks и мониторинг

Health checks для каждого сервиса:

```
app:
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost:8080/health"]
    interval: 30s
```

```
    timeout: 10s
    retries: 3

db:
  healthcheck:
    test: ["CMD-SHELL", "sh -c 'pg_isready -U ${POSTGRES_USER} -d
${POSTGRES_DB}'"]
    interval: 10s
    timeout: 5s
    retries: 5
```

Dozzle для мониторинга логов:

```
dozzle:
  container_name: dozzle
  profiles: [logging, full]
  image: amir20/dozzle:latest
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  expose:
    - "8080"
```

10. Управление данными между сервисами

Database per Service:

- **Backend Service** - использует PostgreSQL для основных данных
- **Email Service** - stateless, не имеет собственной БД
- **Frontend Service** - stateless, хранит только сессионные данные

Shared Database Pattern:

Все сервисы используют общую PostgreSQL БД, но с четким разделением ответственности:

- Backend Service управляет всеми таблицами
- Email Service только читает данные пользователей через Kafka сообщения

11. Конфигурация и секреты

Environment Variables:

```
# Backend
backend/.env.docker:
  DATABASE_URL=postgres://user:pass@db:5432/fitness
  KAFKA_BROKERS=kafka0:29092
  JAEGER_ENDPOINT=http://jaeger:4317

# Email Service
email/.env.docker:
  KAFKA_BROKERS=kafka0:29092
  SMTP_HOST=smtp.gmail.com
  SMTP_PORT=587
```

12. Deployment и масштабирование

Профили для разных окружений:

```
profiles:
  - dev: [database, kafka, tracing, backend, frontend, logging]
  - full: [database, kafka, tracing, backend, frontend, logging, nginx]
  - backend: [database, kafka, tracing, backend, email]
```

Запуск сервисов:

```
# Разработка
docker compose --profile dev up

# Продакшн
docker compose --profile full up

# Только backend сервисы
docker compose --profile backend up
```

[Скриншот архитектуры микросервисов]

[Скриншот Docker Compose сервисов]

[Скриншот Jaeger трассировки]

13. Обработка ошибок между сервисами

Circuit Breaker Pattern:

```
// Retry механизм для Kafka
const retryOptions = {
  retries: 3,
  retryDelayInMs: 1000,
  factor: 2
};
```

Graceful degradation:

```
// Fallback при недоступности email сервиса
if err := s.emailService.SendWelcomeEmail(ctx, email, name); err != nil {
    logger.Errorf("failed to send welcome email: %v", err)
    // Продолжаем работу, email не критичен
}
```

14. API Versioning

URL-based versioning:

```
/v1/users/*
/v1/exercises/*
/v1/workouts/*
```

15. Безопасность между сервисами

Internal network communication:

- Все сервисы в одной Docker сети
- Внешний доступ только через Nginx
- JWT токены для аутентификации

Вывод

Успешно реализована микросервисная архитектура с тремя основными сервисами:

1. **Backend Service** - основная бизнес-логика и API
2. **Email Service** - обработка уведомлений
3. **Frontend Service** - пользовательский интерфейс

Преимущества архитектуры:

- **Независимое развертывание** каждого сервиса
- **Технологическое разнообразие** (Go, TypeScript, React)
- **Масштабируемость** каждого сервиса отдельно
- **Отказоустойчивость** при падении одного сервиса
- **Разделение ответственности** между командами

Сетевое взаимодействие:

- **Синхронное** - HTTP/REST через Nginx
- **Асинхронное** - Kafka для event-driven архитектуры
- **Service Discovery** через Docker Compose
- **Distributed Tracing** с Jaeger
- **Централизованные логи** с Dozzle

Архитектура готова для продакшена и легко масштабируется.