САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №4
Контейнеризация

Выполнил:
Даньшин Семён
К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Реализовать Dockerfile для каждого сервиса. Написать общий docker-compose.yml. Настроить сетевое взаимодействие между сервисами.

Ход работы

1. Контейнеризация Backend Service (Go)

Dockerfile (backend/Dockerfile):

```
ARG GO_VERSION=1.23
FROM --platform=$BUILDPLATFORM golang:${GO_VERSION} AS build
WORKDIR /src

# Кэширование зависимостей
RUN --mount=type=cache,target=/go/pkg/mod/ \
    --mount=type=bind,source=go.sum,target=go.sum \
    --mount=type=bind,source=go.mod,target=go.mod \
    go mod download -x

ARG TARGETARCH

# Сборка основного приложения
RUN --mount=type=cache,target=/go/pkg/mod/ \
    --mount=type=bind,target=. \
    CGO_ENABLED=0 GOARCH=$TARGETARCH go build -o /bin/server ./cmd/workouts

# Сборка утилиты для миграций
RUN --mount=type=cache,target=/go/pkg/mod/ \
    --mount=type=bind,target=. \
    CGO_ENABLED=0 GOARCH=$TARGETARCH go build -o /bin/migrate ./cmd/migrate

# Продакшн образ для основного приложения
FROM alpine:latest AS final

RUN --mount=type=cache,target=/var/cache/apk \
    apk --update add \
        ca-certificates \
        tzdata \
        && \
        update-ca-certificates

ARG UID=10001
RUN adduser \
    --disabled-password \
    --gecos "" \
    --home "/nonexistent" \
    --shell "/sbin/nologin" \
    --no-create-home \
    --uid "${UID}" \
    appuser
USER appuser

COPY --from=build /bin/server /bin/

ENTRYPOINT [ "/bin/server" ]
```

```
# Образ для миграций
FROM alpine:latest AS migrate

RUN --mount=type=cache,target=/var/cache/apk \
    apk --update add \
        ca-certificates \
        tzdata \
        && \
        update-ca-certificates

ARG UID=10001
RUN adduser \
    --disabled-password \
    --gecos "" \
    --home "/nonexistent" \
    --shell "/sbin/nologin" \
    --no-create-home \
    --uid "${UID}" \
    appuser

USER appuser

COPY --from=build /bin/migrate /bin/

ENTRYPOINT [ "/bin/migrate" ]
```

Особенности:

- **Multi-stage build** для оптимизации размера образа
- **Кэширование зависимостей** для ускорения сборки
- **Непривилегированный пользователь** для безопасности
- **Отдельный образ для миграций** БД
- **Поддержка мульти-архитектуры** (ARM64/AMD64)

2. Контейнеризация Email Service (Node.js)

Dockerfile (email/Dockerfile):

```
FROM node:slim AS builder

WORKDIR /app

# Копирование package.json для кэширования зависимостей
COPY package*.json ./

RUN npm install

# Копирование исходного кода и сборка
COPY tsconfig.json ./
COPY src ./src

RUN npm run build

# Продакшн образ
FROM node:slim AS production

WORKDIR /app

# Установка только продакшн зависимостей
```

```
COPY package*.json ./
RUN npm install --omit=dev

# Копирование собранного приложения и шаблонов
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/src/templates ./dist/templates

ENV NODE_ENV=production

CMD ["node", "dist/index.js"]
```

Особенности:

- **Multi-stage build** для уменьшения размера
- **Отдельная установка** dev и prod зависимостей
- **Копирование шаблонов** email для runtime
- **Slim образ** Node.js для минимального размера

3. Контейнеризация Frontend Service (Next.js)

Dockerfile (frontend/Dockerfile):

```
FROM node:slim AS base

WORKDIR /app

# Копирование package.json для кэширования
COPY package*.json ./

# Установка только продакшн зависимостей
RUN npm ci --only=production

# Образ для сборки
FROM node:slim AS build

WORKDIR /app

# Установка всех зависимостей
COPY package*.json ./
RUN npm ci

# Копирование исходного кода и сборка
COPY . .
RUN npm run build

# Финальный продакшн образ
FROM base AS runtime

# Копирование собранного приложения
COPY --from=build /app/.next ./.next
COPY --from=build /app/public ./public

EXPOSE 3000

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

4. Общий Docker Compose файл

Основной compose.yaml:

```
include:
    - database/db-compose.yaml
    - jaeger/jaeger-compose.yaml
    - nginx/nginx-compose.yaml
    - kafka/kafka-compose.yaml

services:
    # Основной backend сервис
    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

    # Инициализация БД (миграции)
    app_init:
        container_name: app_init
        profiles: [backend, full]
        build:
            context: backend
            dockerfile: Dockerfile
            target: migrate
        volumes:
            - ./backend/migrations:/migrations
        env_file:
            - ./backend/.env.docker
        depends_on:
            - db

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

    # Email сервис
```

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

    # Мониторинг логов
    dozzle:
        container_name: dozzle
        profiles: [logging, full]
        environment:
            - DOZZLE_BASE=/logs
        image: amir20/dozzle:latest
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
        expose:
            - "8080"
```

5. Модульная структура compose файлов

Database compose (database/db-compose.yaml):

```
services:
  db:
    image: postgres:15
    container_name: fitness-db
    profiles: [database, full, dev]
    restart: unless-stopped
    volumes:
      - fitness_app_postgres_data:/var/lib/postgresql/data/
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "5430:5432"
    env_file:
      - .env
    healthcheck:
      test: [ "CMD-SHELL", "sh -c 'pg_isready -U ${POSTGRES_USER} -d
${POSTGRES_DB}'" ]
      interval: 10s
      timeout: 5s
      retries: 5

volumes:
  fitness_app_postgres_data:
```

Kafka compose (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
      DYNAMIC_CONFIG_ENABLED: "true"

  kafka0:
    container_name: fitness-kafka
    profiles: [kafka, full, dev]
    image: confluentinc/cp-kafka:7.7.1.arm64
    ports:
      - "9092:9092"
    volumes:
      - fitness_kafka_data:/var/lib/kafka/data
    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_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_CONTROLLER_QUORUM_VOTERS: "1@kafka0:29093"
      KAFKA_PROCESS_ROLES: "broker,controller"
      KAFKA_LOG_DIRS: "/tmp/kraft-combined-logs"
      CLUSTER_ID: 'MkU3OEVBNTcwNTJENDM2Qk'

volumes:
  fitness_kafka_data:
```

Nginx compose (nginx/nginx-compose.yaml):

```
services:
  nginx:
    container_name: nginx
    image: nginx:alpine
    profiles: [frontend, backend, full]
    ports:
      - "8080:80"
    depends_on:
      - app
      - frontend
      - dozzle
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    restart: always
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 1m
      timeout: 10s
      retries: 3
    logging:
      driver: "json-file"
      options:
        max-size: "200k"
        max-file: "10"
```

Jaeger compose (jaeger/jaeger-compose.yaml):

```
services:
  jaeger:
    container_name: jaeger
    profiles: [tracing, full, dev]
    image: jaegertracing/all-in-one
    environment:
        - QUERY_BASE_PATH=/tracing
    expose:
        - "16686"
    ports:
      - "6831:6831/udp"
      - "6832:6832/udp"
      - "5778:5778"
      - "5775:5775/udp"
      - "16686:16686"
      - "4317:4317"
      - "4318:4318"
      - "14250:14250"
      - "14268:14268"
      - "14269:14269"
      - "9411:9411"
```

6. Сетевое взаимодействие

Автоматическая сеть Docker Compose:

```
# Все сервисы автоматически подключаются к сети fitness-trainer_default
# Взаимодействие происходит по именам контейнеров
```

Nginx как reverse proxy:

```
events {}

http {
    upstream backend {
        server app:8080;
    }

    upstream frontend {
        server frontend:3000;
    }

    upstream dozzle {
        server dozzle:8080;
    }

    upstream jaeger {
        server jaeger:16686;
    }

    upstream kafka-ui {
        server fitness-kafka-ui:8080;
    }

    server {
        listen 80;

        # Backend API
```

```
        location /api/ {
            proxy_pass http://backend/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

        # Frontend
        location / {
            proxy_pass http://frontend/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }

        # Мониторинг
        location /logs/ {
            proxy_pass http://dozzle/;
        }

        location /tracing/ {
            proxy_pass http://jaeger/;
        }

        location /kafka-ui/ {
            proxy_pass http://kafka-ui/;
        }
    }
}
```

7. Профили для разных окружений

Доступные профили:

```
# Полное развертывание
docker compose --profile full up

# Только для разработки
docker compose --profile dev up

# Только backend сервисы
docker compose --profile backend up

# Только frontend
docker compose --profile frontend up

# Инфраструктурные сервисы
docker compose --profile database up
docker compose --profile kafka up
docker compose --profile tracing up
```

8. Volumes и персистентность данных

Именованные volumes:

```
volumes:
  fitness_app_postgres_data:    # PostgreSQL данные
  fitness_kafka_data:           # Kafka логи
```

Bind mounts:

```
volumes:
  - ./backend/migrations:/migrations          # Миграции БД
  - ./nginx.conf:/etc/nginx/nginx.conf       # Конфигурация Nginx
  - /var/run/docker.sock:/var/run/docker.sock # Docker socket для Dozzle
```

9. Environment Variables и конфигурация

Backend (.env.docker):

```
DATABASE_URL=postgres://fitness_user:fitness_pass@db:5432/fitness_db?sslmod
e=disable
KAFKA_BROKERS=kafka0:29092
JAEGER_ENDPOINT=http://jaeger:4317
JWT_SECRET=your-secret-key
ENCRYPTION_KEY=your-encryption-key
```

Email Service (.env.docker):

```
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=kafka0:29092
KAFKA_CLIENT_ID=email-service
KAFKA_TOPIC=email-topic
```

10. Health Checks

Проверки состояния сервисов:

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

# Nginx
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost"]
  interval: 1m
  timeout: 10s
  retries: 3
```

11. Makefile для удобства

Makefile команды:

```
compose-local:
    @docker compose --profile dev up --build

compose-local-d:
```

```
    @docker compose --profile dev up --build -d

compose-up:
    @docker compose --profile full up --build

compose-up-d:
    @docker compose --profile full up -d

generate:
    @make -C ./backend generate
    @make -C ./frontend generate

clean:
    @docker compose down -v
    @docker system prune -f
```

12. Логирование

Централизованное логирование:

```
logging:
  driver: "json-file"
  options:
    max-size: "200k"
    max-file: "10"
```

Dozzle для просмотра логов:

```
dozzle:
  container_name: dozzle
  image: amir20/dozzle:latest
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
```

13. Мониторинг и метрики

Jaeger для трассировки:

- Порт 16686 для UI
- Порты 4317/4318 для OTLP
- Все сервисы отправляют traces

Kafka UI:

- Порт 8090 для мониторинга Kafka
- Просмотр топиков и сообщений
- Мониторинг consumer groups

14. Безопасность

Network isolation:

- Все сервисы в одной приватной сети
- Внешний доступ только через Nginx

- Непривилегированные пользователи в контейнерах

Secrets management:

- Environment variables для конфигурации
- .env файлы не коммитятся в git
- Отдельные файлы для разных окружений

15. CI/CD интеграция

GitHub Actions example:

```
- name: Build and test
  run: |
    docker compose --profile backend build
    docker compose --profile backend up -d
    # Запуск тестов
    docker compose down
```

[Скриншот Docker Desktop с запущенными контейнерами]

[Скриншот Docker Compose логов]

[Скриншот Nginx конфигурации]

16. Оптимизация производительности

Layer caching:

- Кэширование Go modules
- Кэширование npm зависимостей
- Multi-stage builds для минимизации размера

Resource limits:

```
deploy:
  resources:
    limits:
      memory: 512M
      cpus: '0.5'
    reservations:
      memory: 256M
      cpus: '0.25'
```

Вывод

Успешно реализована полная контейнеризация приложения:

1. **Dockerfile для каждого сервиса** с оптимизацией размера и безопасности
2. **Модульная структура Docker Compose** для гибкого управления
3. **Сетевое взаимодействие** через Docker networks и Nginx proxy
4. **Профили окружений** для разработки и продакшена
5. **Health checks** для мониторинга состояния
6. **Централизованное логирование** и мониторинг
7. **Персистентность данных** через volumes
8. **Безопасность** с изоляцией сетей и непривилегированными пользователями

Преимущества контейнеризации:

- **Консистентность** между окружениями
- **Простота развертывания** одной командой
- **Масштабируемость** каждого сервиса
- **Изоляция** сервисов друг от друга
- **Легкость мониторинга** и отладки

Приложение готово для развертывания в любом окружении с поддержкой Docker.