САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая/Лабораторная работа

Выполнил:

Габов Михаил

К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Цель работы: Изолировать каждый сервис приложения в отдельный контейнер, настроить их совместную работу и обеспечить простоту развертывания с помощью Docker.

1. Реализация Dockerfile для сервисов

Для каждого микросервиса (api-gateway, auth-users-service, recipes-service, interactions-service) был подготовлен Dockerfile для сборки его образа. Все сервисы, написанные на Node.js и TypeScript, используют схожую структуру Dockerfile, основанную на многоэтапной сборке для оптимизации размера конечного образа и повышения безопасности.

Пример Dockerfile для сервисов:

TypeScript в JavaScript
```
FROM node:18-alpine AS builder

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

FROM node:18-alpine

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install --omit=dev
```

```
COPY --from=builder /usr/src/app/dist ./dist

EXPOSE 8001

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

Ключевые моменты Dockerfile:

- **Многоэтапная сборка (FROM ... AS builder):** Позволяет использовать один контейнер со всеми инструментами для сборки (typescript, ts-node), а затем скопировать только необходимые артефакты (скомпилированный JS-код и node_modules) в чистый, легковесный контейнер. Это значительно уменьшает размер итогового образа.
- **Базовый образ node:18-alpine:** Выбран как один из самых легковесных и безопасных образов для запуска Node.js приложений.
- **Кэширование слоев:** Копирование package*.json и установка зависимостей выполняются отдельными шагами перед копированием исходного кода. Это позволяет Docker кэшировать слой с установленными модулями, и npm install не будет выполняться каждый раз при изменении кода, что ускоряет сборку.
- **--omit=dev:** При установке зависимостей в конечном образе мы исключаем пакеты, нужные только для разработки, что также уменьшает размер.

Аналогичный Dockerfile, с небольшими изменениями в портах и командах запуска, был создан для сервиса api-gateway.

2. Общий файл docker-compose.yml

Docker-compose.yml определяет все компоненты системы, их зависимости, переменные окружения и сетевые настройки.

Содержимое файла docker-compose.yml:

```
version: '3.8'
```

```
services:
  api-gateway:
    build: ./services/api-gateway
    container_name: api-gateway
    ports:
      - "8000:8000"
    environment:
      - AUTH_USERS_SERVICE_URL=http://auth-users-service:8001
      - RECIPES_SERVICE_URL=http://recipes-service:8002
      - INTERACTIONS_SERVICE_URL=http://interactions-service:8003
    depends_on:
      - auth-users-service
      - recipes-service
      - interactions-service

  auth-users-service:
    build: ./services/auth-users
    container_name: auth-users-service
    ports:
      - "8001:8001"
    environment:
      - DB_HOST=postgres-db
      - DB_PORT=5432
      - DB_USERNAME=user
      - DB_PASSWORD=password
      - DB_DATABASE=auth_db
      - RABBITMQ_HOST=rabbitmq-broker
    depends_on:
      - postgres-db
      - rabbitmq

  recipes-service:
```

```
  build: ./services/recipes
  container_name: recipes-service
  ports:
    - "8002:8002"
  environment:
    - DB_HOST=postgres-db
    - DB_PORT=5432
    - DB_USERNAME=user
    - DB_PASSWORD=password
    - DB_DATABASE=recipes_db
    - RABBITMQ_HOST=rabbitmq-broker
  depends_on:
    - postgres-db
    - rabbitmq

interactions-service:
  build: ./services/interactions
  container_name: interactions-service
  ports:
    - "8003:8003"
  environment:
    - DB_HOST=postgres-db
    - DB_PORT=5432
    - DB_USERNAME=user
    - DB_PASSWORD=password
    - DB_DATABASE=interactions_db
    - RABBITMQ_HOST=rabbitmq-broker
  depends_on:
    - postgres-db
    - rabbitmq

postgres-db:
  image: postgres:16-alpine
```

```
  container_name: postgres-db
  environment:
    - POSTGRES_USER=user
    - POSTGRES_PASSWORD=password
  volumes:
    - postgres-data:/var/lib/postgresql/data
    - ./init-db.sh:/docker-entrypoint-initdb.d/init-db.sh
  ports:
    - "5432:5432"

rabbitmq:
  image: rabbitmq:3-management-alpine
  container_name: rabbitmq-broker
  hostname: rabbitmq-broker
  ports:
    - "5672:5672"
    - "15672:15672"

postgres-data:
```

Ключевые моменты docker-compose.yml:

- **build: ./services/...**: Указывает Docker Compose, что образ для сервиса нужно собирать из Dockerfile, находящегося в указанной директории.
- **image: ...**: Для PostgreSQL и RabbitMQ используются готовые образы с Docker Hub.
- **environment**: Через этот блок в каждый контейнер передаются необходимые переменные окружения: адреса базы данных и RabbitMQ, учетные данные и т.д.
- **ports**: Пробрасывает порты из контейнера на хост-машину. Внешнему миру доступен только api-gateway (порт 8000), база данных (5432 для отладки) и UI RabbitMQ (15672).
- **volumes**: postgres-data используется для сохранения данных PostgreSQL,

чтобы они не терялись при остановке или удалении контейнера. Также монтируется скрипт init-db.sh для автоматического создания баз данных при первом запуске.

- **depends_on**: Определяет порядок запуска контейнеров. Например, сервисы приложений стартуют только после запуска postgres-db и rabbitmq.

3. Настройка сетевого взаимодействия

Docker Compose по умолчанию создает единую виртуальную сеть для всех сервисов, определенных в файле docker-compose.yml. Это позволяет контейнерам общаться друг с другом, используя имена сервисов в качестве хост-имен.

- **Внутренняя коммуникация:**
  - api-gateway обращается к другим сервисам по их именам: http://auth-users-service:8001, http://recipes-service:8002 и т.д. Эти адреса передаются через переменные окружения.
  - Все сервисы приложений (auth-users, recipes, interactions) подключаются к базе данных по адресу postgres-db:5432 и к брокеру сообщений по адресу rabbitmq-broker:5672.
- **Внешняя коммуникация:**
  - Единственной точкой входа для внешних запросов является api-gateway на порту 8000. Он принимает запросы от пользователей и проксирует их к соответствующим внутренним сервисам.
  - Для отладки и администрирования открыты порты базы данных (5432) и веб-интерфейса RabbitMQ (15672).

Заключение

В рамках данной работы была выполнена контейнеризация приложения. Для каждого сервиса подготовлен Dockerfile