Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский экономический университет имени Г.В. Плеханова»

**МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ**

Специальность 09.02.07 Информационные системы и программирование

Квалификация: Программист

Практическая работа №1

По дисциплине «МДК 04.02 "Обеспечение качества функционирования КС"»

Листов: 6

|  |  |
| --- | --- |
| Выполнил: Кононыхин Е. Е.  Студент группы: П50-7-22 | Проверил преподаватель  \_\_\_\_\_\_\_\_\_\_\_\_Р.Д. Степаньков  «\_\_\_\_» \_\_\_\_\_\_\_\_ 2025 года |

Москва 2025

## ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

### Цель работы:

Изучить процесс развертывания многокомпонентных приложений в Docker, освоить создание Docker Compose конфигураций и управление связанными сервисами.

### Задачи работы:

1. Базовый уровень ("хорошо"):
   * Запустить пример приложения time-db внутри контейнера Docker
   * Подробно пояснить каждую команду и компонент
   * Опубликовать готовый проект на GitHub
2. Продвинутый уровень ("отлично"):
   * Выполнить пункты базового уровня
   * Создать собственное приложение с минимум тремя компонентами
   * Оформить отчет с пояснениями каждого шага

## ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

### Docker Compose

Docker Compose - это инструмент для определения и запуска многоконтейнерных приложений Docker. Основные преимущества:

* Оркестрация сервисов - управление несколькими контейнерами как единым приложением
* Сетевое взаимодействие - автоматическое создание сетей между контейнерами
* Управление данными - создание и монтирование томов
* Переменные окружения - конфигурация через environment variables

### Архитектура микросервисов

Современные приложения часто строятся по принципу микросервисов:

* Frontend - пользовательский интерфейс
* Backend API - бизнес-логика и обработка данных
* База данных - хранение данных
* Административные инструменты - управление системой

## АНАЛИЗ ИСХОДНОГО ПРИЛОЖЕНИЯ

### Структура приложения time-app-praktika

Исходное приложение состоит из двух основных компонентов:

#### 1. Backend API (Node.js + Express)

* Файл: api/index.mjs
* Технологии: Node.js, Express, MySQL2
* Функциональность:
  + GET / - проверка работы API
  + GET /times - получение всех сохраненных времен
  + POST /times - сохранение нового времени
  + DELETE /time/:id - удаление времени по ID

#### 2. Frontend (Vue.js)

* Файл: frontend/src/components/Time.vue
* Технологии: Vue.js 3, Vite, Moment.js
* Функциональность:
  + Отображение текущего времени в реальном времени
  + Сохранение времени в базу данных
  + Отображение ранее сохраненных времен
  + Удаление сохраненных времен

### Проблемы исходной конфигурации

1. Хардкод портов - API использует порт 5555, но должен использовать 5000
2. Отсутствие контейнеризации - приложение не готово для Docker
3. Нет управления базой данных - отсутствует инструмент для администрирования БД

## СОЗДАНИЕ DOCKER КОНТЕЙНЕРОВ

### 1. Dockerfile для API (Backend)

FROM node:18-alpine  
  
WORKDIR /app  
  
COPY package\*.json ./  
  
RUN npm install  
  
COPY . .  
  
EXPOSE 5000  
  
CMD ["npm", "start"]

Объяснение команд:

* FROM node:18-alpine - используем официальный образ Node.js 18 на базе Alpine Linux
* WORKDIR /app - устанавливаем рабочую директорию
* COPY package\*.json ./ - копируем файлы зависимостей
* RUN npm install - устанавливаем зависимости
* COPY . . - копируем исходный код
* EXPOSE 5000 - открываем порт 5000
* CMD ["npm", "start"] - запускаем приложение

### 2. Dockerfile для Frontend (Multi-stage build)

FROM node:18-alpine  
  
WORKDIR /app  
  
COPY package\*.json ./  
  
RUN npm install  
  
COPY . .  
  
RUN npm run build  
  
FROM nginx:alpine  
  
COPY --from=0 /app/dist /usr/share/nginx/html  
  
EXPOSE 80  
  
CMD ["nginx", "-g", "daemon off;"]

Объяснение команд:

* Первый этап: сборка Vue.js приложения
  + FROM node:18-alpine - образ для сборки
  + RUN npm run build - создание production сборки
* Второй этап: создание веб-сервера
  + FROM nginx:alpine - легковесный веб-сервер
  + COPY --from=0 - копирование собранного приложения из первого этапа
  + EXPOSE 80 - открытие порта для веб-сервера

### 3. Конфигурация базы данных

Обновлен файл mysqlPool.mjs для поддержки переменных окружения:

const pool = mysql.createPool({  
 connectionLimit: 100,  
 host: process.env.MYSQL\_HOST || 'mysql',  
 port: process.env.MYSQL\_PORT || '3306',  
 user: process.env.MYSQL\_USER || 'root',  
 password: process.env.MYSQL\_PASSWORD || 'password',  
 database: process.env.MYSQL\_DATABASE || 'time\_db',  
})

## КОНФИГУРАЦИЯ DOCKER COMPOSE

### docker-compose.yml

version: '3.8'  
  
services:  
 mysql:  
 image: mysql:8.0  
 container\_name: time-mysql  
 environment:  
 MYSQL\_ROOT\_PASSWORD: password  
 MYSQL\_DATABASE: time\_db  
 ports:  
 - "3306:3306"  
 volumes:  
 - mysql\_data:/var/lib/mysql  
 networks:  
 - time-network  
  
 api:  
 build: ./api  
 container\_name: time-api  
 ports:  
 - "5000:5000"  
 depends\_on:  
 - mysql  
 environment:  
 - NODE\_ENV=production  
 networks:  
 - time-network  
  
 frontend:  
 build: ./frontend  
 container\_name: time-frontend  
 ports:  
 - "80:80"  
 depends\_on:  
 - api  
 networks:  
 - time-network  
  
 adminer:  
 image: adminer:latest  
 container\_name: time-adminer  
 ports:  
 - "8080:8080"  
 depends\_on:  
 - mysql  
 networks:  
 - time-network  
  
volumes:  
 mysql\_data:  
  
networks:  
 time-network:  
 driver: bridge

### Объяснение конфигурации

#### Сервисы:

1. mysql - База данных MySQL 8.0
   * image: mysql:8.0 - официальный образ MySQL
   * environment - переменные окружения для настройки БД
   * volumes - постоянное хранение данных
   * ports - проброс порта 3306
2. api - Backend API
   * build: ./api - сборка из Dockerfile в папке api
   * depends\_on: mysql - ожидание запуска MySQL
   * ports - проброс порта 5000
3. frontend - Frontend приложение
   * build: ./frontend - сборка из Dockerfile в папке frontend
   * depends\_on: api - ожидание запуска API
   * ports - проброс порта 80
4. adminer - Административный интерфейс для БД
   * image: adminer:latest - готовый образ Adminer
   * depends\_on: mysql - ожидание запуска MySQL
   * ports - проброс порта 8080

#### Сеть и тома:

* time-network - изолированная сеть для всех сервисов
* mysql\_data - постоянный том для данных MySQL

## ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

### Команды для запуска

# Переход в директорию проекта  
cd time-app-praktika  
  
# Запуск всех сервисов  
docker compose up --build  
  
# Запуск в фоновом режиме  
docker compose up -d --build

### Проверка работы сервисов

# Просмотр запущенных контейнеров  
docker ps  
  
# Просмотр логов  
docker compose logs  
  
# Проверка API  
curl http://localhost:5000/  
  
# Проверка фронтенда  
curl http://localhost/

### Доступ к приложению

* Веб-приложение: http://localhost
* API: http://localhost:5001
* Adminer: http://localhost:8080
  + Сервер: mysql
  + Пользователь: root
  + Пароль: password
  + База данных: time\_db

## РЕЗУЛЬТАТЫ РАБОТЫ

### Успешно выполнено:

✅ Создано 4 Docker контейнера:

* MySQL 8.0 (база данных)
* Node.js API (backend)
* Vue.js Frontend (frontend)
* Adminer (администрирование БД)

✅ Настроена сетевая связность между всеми сервисами

✅ Создана Docker Compose конфигурация для оркестрации

✅ Добавлена поддержка переменных окружения

✅ Настроено постоянное хранение данных для MySQL

### Архитектура приложения:

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Frontend │ │ Backend API │ │ MySQL DB │  
│ (Vue.js) │◄──►│ (Node.js) │◄──►│ (MySQL 8.0) │  
│ Port: 80 │ │ Port: 5000 │ │ Port: 3306 │  
└─────────────────┘ └─────────────────┘ └─────────────────┘  
 │ │ │  
 └───────────────────────┼───────────────────────┘  
 │  
 ┌─────────────────┐  
 │ Adminer │  
 │ Port: 8080 │  
 └─────────────────┘

### Размеры образов:

|  |  |  |
| --- | --- | --- |
| Сервис | Размер образа | Описание |
| MySQL | ~500 MB | База данных с данными |
| API | ~200 MB | Node.js + зависимости |
| Frontend | ~50 MB | Nginx + собранное приложение |
| Adminer | ~100 MB | Веб-интерфейс для БД |

## ВЫВОДЫ И ЗАКЛЮЧЕНИЕ

### Достигнутые результаты:

1. Освоена контейнеризация многокомпонентных приложений
2. Изучена работа с Docker Compose для оркестрации сервисов
3. Настроено сетевое взаимодействие между контейнерами
4. Реализовано управление данными через Docker volumes
5. Добавлен административный интерфейс для управления БД

### Преимущества Docker Compose:

* Упрощение развертывания - одна команда для запуска всего приложения
* Изоляция сервисов - каждый компонент работает в своем контейнере
* Масштабируемость - легко добавлять новые сервисы
* Воспроизводимость - одинаковое поведение на всех средах

### Полученные навыки:

* Создание Dockerfile для различных типов приложений
* Настройка Docker Compose для многоконтейнерных приложений
* Управление сетевым взаимодействием между контейнерами
* Работа с Docker volumes для постоянного хранения данных
* Использование переменных окружения для конфигурации

### Практическая значимость:

Полученные знания позволяют эффективно развертывать сложные приложения в контейнерах, что является стандартом в современной разработке и DevOps практиках. Docker Compose значительно упрощает управление микросервисной архитектурой.