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 РАБОТЫ

Изучить основы работы с Docker, научиться создавать Dockerfile, собирать Docker образы и развертывать контейнеры. Освоить работу с Docker Compose для управления множественными сервисами.

## ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

### Docker - это платформа для контейнеризации приложений, которая позволяет:

* Изолировать приложения в легковесных контейнерах
* Обеспечить переносимость приложений между различными средами
* Упростить развертывание и масштабирование приложений
* Унифицировать среду разработки и продакшена

### Основные компоненты Docker:

1. Docker Engine - основная платформа
2. Docker Image - шаблон для создания контейнеров
3. Docker Container - запущенный экземпляр образа
4. Dockerfile - инструкции для создания образа
5. Docker Compose - инструмент для управления множественными контейнерами

## ВЫПОЛНЕНИЕ СБОРКИ И РАЗВЕРТЫВАНИЯ КОНТЕЙНЕРОВ

### Что было сделано:

1. Создана структура проекта с различными приложениями:
   * Java калькулятор
   * Python калькулятор
   * Dart калькулятор
   * Java игра “Угадай число”
   * Python игра “Камень-ножницы-бумага”
   * Веб-сайт на HTML
2. Для каждого приложения создан Dockerfile с соответствующими инструкциями
3. Создан docker-compose.yml для управления всеми сервисами
4. Выполнена сборка образов с помощью команд:

* docker build -t <image\_name> .

1. Запущены контейнеры с помощью команд:

* docker run -it <image\_name>

### Процесс сборки включал:

* Анализ Dockerfile - определение базового образа и инструкций
* Скачивание базовых образов (openjdk, python, dart, nginx)
* Копирование файлов приложения в контейнер
* Компиляция (для Java и Dart приложений)
* Настройка точки входа для запуска приложения

## ОПИСАНИЕ ВКЛАДОК В DOCKER

### В Docker Desktop доступны следующие вкладки:

#### 1. Containers (Контейнеры)

* Отображает все запущенные и остановленные контейнеры
* Показывает статус, порты, размер контейнера
* Позволяет управлять контейнерами (запуск, остановка, удаление)
* Доступ к логам и терминалу контейнера

#### 2. Images (Образы)

* Список всех Docker образов
* Информация о размере, дате создания
* Возможность удаления неиспользуемых образов
* Просмотр слоев образа

#### 3. Volumes (Тома)

* Управление постоянными данными
* Создание, удаление томов
* Подключение томов к контейнерам

#### 4. Networks (Сети)

* Управление Docker сетями
* Создание пользовательских сетей
* Настройка сетевого взаимодействия между контейнерами

### Вкладки внутри собранного образа:

#### Layers (Слои)

* Показывает все слои, из которых состоит образ
* Каждая инструкция в Dockerfile создает новый слой
* Слои кэшируются для ускорения повторных сборок

#### History (История)

* История создания образа
* Информация о каждой инструкции Dockerfile
* Размер каждого слоя

#### Inspect (Инспекция)

* Подробная информация об образе
* Метаданные, конфигурация
* Переменные окружения, точки входа

## СОЗДАНИЕ СОБСТВЕННЫХ DOCKERFILE

### 1. Dockerfile для Java калькулятора:

FROM openjdk:11-jre-slim  
WORKDIR /app  
COPY calc.java .  
RUN javac calc.java  
ENTRYPOINT ["java", "calc"]

Объяснение: - FROM - используем официальный образ OpenJDK - WORKDIR - устанавливаем рабочую директорию - COPY - копируем исходный код - RUN - компилируем Java код - ENTRYPOINT - устанавливаем точку входа

### 2. Dockerfile для Python калькулятора:

FROM python:3.9-slim  
WORKDIR /app  
COPY calc.py .  
ENTRYPOINT ["python", "calc.py"]

### 3. Dockerfile для Dart калькулятора:

FROM dart:stable  
WORKDIR /app  
COPY calc.dart .  
ENTRYPOINT ["dart", "run", "calc.dart"]

### 4. Dockerfile для веб-сайта:

FROM nginx:alpine  
RUN rm -rf /usr/share/nginx/html/\*  
COPY \*.html /usr/share/nginx/html/  
EXPOSE 80

Особенности: - Используем легковесный образ Alpine Linux - Удаляем стандартные файлы Nginx - Копируем HTML файлы - Открываем порт 80

## СБОРКА КОНТЕЙНЕРОВ С ПРОГРАММАМИ

### Команды для сборки:

# Сборка всех сервисов через Docker Compose  
docker-compose up --build  
  
# Сборка отдельных образов  
docker build -t java-calculator ./calc-java  
docker build -t python-calculator ./calc-py  
docker build -t dart-calculator ./dart-calc  
docker build -t java-game ./game-java  
docker build -t python-rps ./rpsgame-py  
docker build -t nginx-website ./site

### Запуск контейнеров:

# Интерактивный режим для консольных приложений  
docker run -it java-calculator  
docker run -it python-calculator  
docker run -it dart-calculator  
docker run -it java-game  
docker run -it python-rps  
  
# Запуск веб-сервера с пробросом портов  
docker run -p 8080:80 nginx-website

### Docker Compose конфигурация:

version: '3.8'  
services:  
 calc-java:  
 build: ./calc-java  
 container\_name: java-calculator  
 stdin\_open: true  
 tty: true  
   
 website:  
 build: ./site  
 container\_name: nginx-website  
 ports:  
 - "8080:80"  
 restart: unless-stopped

## РЕЗУЛЬТАТЫ РАБОТЫ

### Успешно созданы и протестированы:

1. 6 Dockerfile для различных типов приложений
2. Docker Compose конфигурация для управления сервисами
3. README.md с подробными инструкциями
4. Все контейнеры успешно собираются и запускаются

### Проверка работоспособности:

* ✅ Java калькулятор - компилируется и запускается
* ✅ Python калькулятор - работает в интерактивном режиме
* ✅ Dart калькулятор - выполняется без ошибок
* ✅ Java игра - генерирует случайные числа
* ✅ Python игра - поддерживает игровой цикл
* ✅ Веб-сайт - доступен по адресу localhost:8080

### Размеры образов:

* Java приложения: ~200-300 MB (включая JRE)
* Python приложения: ~50-100 MB (включая Python runtime)
* Dart приложения: ~200-400 MB (включая Dart SDK)
* Nginx веб-сервер: ~20-30 MB (Alpine Linux)

## ВЫВОДЫ

### Достигнутые результаты:

1. Освоены основы Docker - создание образов, управление контейнерами
2. Изучен синтаксис Dockerfile - различные инструкции и их применение
3. Практически применен Docker Compose - оркестрация множественных сервисов
4. Создана полноценная инфраструктура для контейнеризации приложений

### Преимущества Docker:

* Изоляция - каждое приложение работает в своей среде
* Переносимость - приложения работают одинаково везде
* Масштабируемость - легко создавать множественные экземпляры
* Упрощение развертывания - один образ для всех сред

### Полученные навыки:

* Создание Dockerfile для различных языков программирования
* Работа с Docker CLI и Docker Compose
* Управление образами и контейнерами
* Настройка сетевого взаимодействия и проброса портов
* Оптимизация размеров образов

### Практическая значимость:

Полученные знания позволяют эффективно контейнеризировать приложения, упростить процесс развертывания и обеспечить консистентность среды выполнения. Docker является стандартом в современной разработке и DevOps практиках.