МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3

по дисциплине

«Технологии виртуализации клиент-серверных приложений»

Выполнил студент группы ИКБО-01-22 Прокопчук Р.О.

Принял преподаватель кафедры ИиППО Волков М.Ю.

Практические работы выполнены «__»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва
2025

Теоретическое введение

Docker — это программная платформа для разработки, доставки и запуска контейнерных приложений. Он позволяет создавать контейнеры, автоматизировать их запуск и развертывание, управляет жизненным циклом. С помощью Docker можно запускать множество контейнеров на одной хостмашине.

Контейнеризация — это способ упаковки приложения и всех его зависимостей в один образ, который запускается в изолированной среде, не влияющей на основную операционную систему.

Контейнер — базовая единица программного обеспечения, покрывающая код и все его зависимости для обеспечения запуска приложения прозрачно, быстро и надежно независимо от окружения. Контейнер Docker может быть создан с использованием образа Docker. Это исполняемый пакет программного обеспечения, содержащий все необходимое для запуска приложения, например, системные программы, библиотеки, код, среды исполнения и настройки.

Docker-образ — шаблон для создания Docker-контейнеров. Представляет собой исполняемый пакет, содержащий все необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации.

Docker-образ состоит из слоев. Каждое изменение записывается в новый слой. При загрузке или скачивании Docker-образа, операции производятся только с теми слоями, которые были изменены. Слои исходного Docker-образа являются общими между всеми его версиями и не дублируются.

Тома Docker — это способ создания постоянного хранилища для контейнеров Docker. Тома Docker не привязаны к времени жизни контейнера, поэтому сделанные в них записи не исчезнут, как это произойдет с контейнером. Они также могут быть повторно подключены к одному или к нескольким контейнерам, чтобы можно было обмениваться данными и подключать новые контейнеры к существующему хранилищу. Тома Docker

работают путем создания каталога на главной машине и последующего монтирования этого каталога в контейнер (или в несколько контейнеров). Этот каталог существует вне многослойного образа, который обычно содержит контейнер Docker, поэтому он не подчиняется тем же правилам (только для чтения и т. д.).

Часто используемые команды Docker:

- docker push: Закачать репозиторий или образ в Registry;
- docker run: Запустить команду в новом контейнере;
- docker pull: Скачать репозиторий или образ из Registry;
- docker start: Запустить один или несколько контейнеров;
- docker stop: Остановить один или несколько контейнеров;
- docker search: Поиск образа на DockerHub;
- docker commit: Сохранить изменения в новый образ.
- docker -version: узнать установленную версию Docker;
- docker ps: перечислить все запущенные контейнеры вместе с дополнительной информацией о них;
- docker ps -a: перечислить все контейнеры, включая остановленные, вместе с дополнительной информацией о них;
- docker exec: войти в контейнер и выполнить в нем команду;
- docker build: собрать образ из Dockerfile;
- docker rm: удалить контейнер с указанным идентификатором;
- docker rmi: удалить образ с указанным идентификатором;
- docker info: получить расширенную информацию об установленном Docker, например, сколько запущено контейнеров, образов, версию ядра, доступную оперативную память и т.п.;
- docker cp: сохранить файл из контейнера в локальную систему;
- docker history: показать историю образа с указанным именем.

Все возможные состояния контейнера Docker:

- Created — контейнер создан, но не активен.
- Restarting — контейнер в процессе перезапуска.

● Running — контейнер работает.

● Paused — контейнер приостановлен.

● Exited — контейнер закончил свою работу.

● Dead — контейнер, который сервис попытался остановить, но не смог.

Dockerfile содержит инструкции для сборки образов, которые передаются в Docker. Также его можно описать как текстовый документ, содержащий все возможные команды, с помощью которых пользователь, последовательно их запуская, может собрать образ.

Swarm Mode — это встроенная система оркестровки Docker для масштабирования контейнеров в кластере физических машин. Несколько независимых клиентов, на которых работает Docker Engine, объединяют свои ресурсы, образуя рой.

Эта функция поставляется в комплекте с Docker и включает все необходимое для развертывания приложений на узлах. Swarm Mode имеет декларативную модель масштабирования, в которой вы указываете количество необходимых реплик. Менеджер роя принимает меры, чтобы сопоставить фактическое количество реплик вашему запросу, создавая и уничтожая контейнеры по мере необходимости.

Постановка задачи

Задание 1

Создать один или несколько Dockerfile, в которых каждая из нижеприведенных команд будет использована хотя бы 1 раз:

— FROM;

— RUN;

— LABEL;

— CMD;

— EXPOSE;

— ENV;

— ADD;

— COPY;

— ENTRYPOINT;

— VOLUME;

— USER;

— WORKDIR;

— ONBUILD.

Задание 2

- Составлен Dockerfile с веб-приложением.
- Параметры веб приложения:
    - Написано с помощью фреймворка Spring Boot;
    - Взаимодействует с СУБД PostgreSQL;
    - Реализованы следующие эндпоинты:
        - добавление элемента,
        - вывод списка всех элементов,
        - получение фото герба РТУ МИРЭА.
- При сборке проекта с помощью консольной утилиты wget скачан файл по адресу

https://www.mirea.ru/upload/medialibrary/80f/MIREA_Gerb_Colour.png. Файл должен быть доступен по эндпоинту веб-сервиса.

- Порт СУБД должен быть получен из переменной окружения, указанной в Dockerfile.
- Dockerfile должен содержать 2 стадии: сборка и запуск jar файла, стадии должны быть изолированы.
- В LABEL должны быть указаны ФИО и группа студента.
- По завершению запуска сервиса произведен вывод строки с вашим ФИО “Сборка и запуск произведены. Автор: {ФИО студента}” с помощью команды ONBUILD.
- Docker образ загружен в DockerHub.

Ход работы

Dockerfile со всеми основными командами для задания 1 представлен на рисунке 1.

```
FROM alpine:latest

LABEL author="Prokopchuk Roman" group="IKBO-01-22"

ENV APP_VERSION=1.0

WORKDIR /app

ADD https://raw.githubusercontent.com/docker-library/hello-world/master/hello.c .

COPY file.txt .

RUN adduser -D remsely

USER remsely

VOLUME /app/data

EXPOSE 8080

ENTRYPOINT ["/bin/sh", "-c"]

CMD ["echo App ${APP_VERSION} started."]

ONBUILD RUN echo "Creating new image based on my image..."
```

Рисунок 1 – Dockerfile со всеми основными командами

Код приложения, написанного для задания 2, представлен на рисунках 2-8.

```
package edu.mirea.remsely.csavt.practice3.mirealogodownloader.entity

import jakarta.persistence.Entity
import jakarta.persistence.GeneratedValue
import jakarta.persistence.GenerationType
import jakarta.persistence.Id

@Entity 7 Usages  Remsely
open data class Item(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long = 0,
    val name: String
)
```

Рисунок 2 – Класс сущности Item

```
package edu.mirea.remsely.csavt.practice3.mirealogodownloader.repository

import edu.mirea.remsely.csavt.practice3.mirealogodownloader.entity.Item
import org.springframework.data.jpa.repository.JpaRepository

interface ItemRepository : JpaRepository<Item, Long>  2 Usages  Remsely
```

Рисунок 3 – Репозиторий сущности Item

```
package edu.mirea.remsely.csavt.practice3.mirealogodownloader.service

import edu.mirea.remsely.csavt.practice3.mirealogodownloader.dto.ItemDto
import edu.mirea.remsely.csavt.practice3.mirealogodownloader.mapper.toDto
import edu.mirea.remsely.csavt.practice3.mirealogodownloader.mapper.toEntity
import edu.mirea.remsely.csavt.practice3.mirealogodownloader.repository.ItemRepository
import org.springframework.stereotype.Service

@Service  2 Usages  Remsely
open class ItemService(
    private val itemRepository: ItemRepository
) {
    open fun saveItem(item: ItemDto): ItemDto = itemRepository.save( entity = item.toEntity()).toDto()

    open fun getAllItems(): List<ItemDto> = itemRepository.findAll().toDto()  1 Usage  Remsely
}
```

Рисунок 4 – Сервис для работы с item

```
package edu.mirea.remsely.csavt.practice3.mirealogodownloader.service

import org.springframework.core.io.ClassPathResource
import org.springframework.stereotype.Service

@Service  2 Usages  Remsely
open class MireaLogoProvider {
    open fun getLogo(): ByteArray? {  1 Usage  Remsely
        val resource = ClassPathResource( path = "static/MIREA_Gerb_Colour.png")
        return if (resource.exists()) {
            resource.inputStream.readBytes()
        } else {
            null
        }
    }
}
```

Рисунок 5 – Сервис для получения логотипа из ресурсов

```kotlin
package edu.mirea.remsely.csavt.practice3.mirealogodownloader.controller

import edu.mirea.remsely.csavt.practice3.mirealogodownloader.dto.ItemDto
import edu.mirea.remsely.csavt.practice3.mirealogodownloader.service.ItemService
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.PostMapping
import org.springframework.web.bind.annotation.RequestBody
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController  Remsely
@RequestMapping( ...value = "/items")
open class ItemController(
    private val itemService: ItemService,
) {
    @PostMapping  Remsely
    open fun addItem(@RequestBody item: ItemDto): ItemDto = itemService.saveItem(item)

    @GetMapping  Remsely
    open fun getAllItems(): List<ItemDto> = itemService.getAllItems()
}
```

Рисунок 6 – Контроллер для работы с item

```kotlin
package edu.mirea.remsely.csavt.practice3.mirealogodownloader.controller

import edu.mirea.remsely.csavt.practice3.mirealogodownloader.service.MireaLogoProvider
import org.springframework.http.MediaType
import org.springframework.http.ResponseEntity
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController  Remsely
@RequestMapping( ...value = "/mirea/logo")
open class LogoController(
    private val logoProvider: MireaLogoProvider
) {
    @GetMapping(produces = [MediaType.IMAGE_PNG_VALUE])  Remsely
    open fun getCrestImage(): ResponseEntity<ByteArray> =
        logoProvider.getLogo().let { logoBytes ->
            if (logoBytes != null) {
                ResponseEntity.ok()
                    .contentType( contentType = MediaType.IMAGE_PNG)
                    .body( body = logoBytes)
            } else {
                ResponseEntity.notFound().build()
            }
        }
}
```

Рисунок 7 – Контроллер для работы с логотипом

```yaml
spring:
  application:
    name: mirea-logo-downloader

  datasource:
    url: jdbc:postgresql://${DB_HOST:localhost}:${DB_PORT:5432}/${DB_NAME:postgres}
    username: ${DB_USER:postgres}
    password: ${DB_PASSWORD:postgres}
    driver-class-name: org.postgresql.Driver

  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
```

Рисунок 8 – Настройки Spring Boot приложения

Dockerfile, реализованный для задания 2 представлен на рисунке 9.

```dockerfile
FROM gradle:8.14.3-jdk21 AS build

WORKDIR /app

COPY . .

RUN apt-get update && apt-get install -y wget && \
    wget -P src/main/resources/static https://www.mirea.ru/upload/medialibrary/80f/MIREA_Gerb_Colour.png

RUN gradle build -x test --no-daemon

FROM openjdk:21-slim

LABEL author="Прокопчук Р.О." group="ИКБО-01-22"

WORKDIR /app

ENV DB_PORT=5432

COPY --from=build /app/build/libs/*.jar app.jar

EXPOSE 8080

ONBUILD RUN echo "Сборка и запуск произведены. Автор: Прокопчук Р.О."

ENTRYPOINT ["java", "-jar", "app.jar"]
```

Рисунок 9 – Dockerfile для задания 2

Вывод

В результате выполнения данной практической работы были созданы Dockerfile-ы со всеми основными командами и Dockerfile для запуска Spring Boot приложения.