|  |  |  |
| --- | --- | --- |
|  | Изображение выглядит как зарисовка, рисунок, символ, корона  Автоматически созданное описание |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №3**

по дисциплине «Технологии виртуализации клиент-серверных приложений»

**Тема практической работы: «Создание образов Docker»**

**Студент группы** ИКБО-20-21 Фомичев Р.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** старший преподаватель Волков М.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Москва 2024

**СОДЕРЖАНИЕ**

[ЦЕЛЬ РАБОТЫ 3](#_Toc177154413)

[ЗАДАНИЕ №1 4](#_Toc177154414)

[ЗАДАНИЕ №2 7](#_Toc177154415)

[ВЫВОД 11](#_Toc177154416)

[ОТВЕТЫ НА ВОПРОСЫ К ПРАКТИЧЕСКОЙ РАБОТЕ 12](#_Toc177154417)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14](#_Toc177154418)

ЦЕЛЬ РАБОТЫ

Освоить основные инструкции Dockerfile и научиться их применять на практике. Получить опыт использования различных команд Docker, таких как FROM, RUN, LABEL, CMD, EXPOSE, ENV, ADD, COPY, ENTRYPOINT, VOLUME, USER, WORKDIR и ONBUILD.

Применить полученные знания для создания более сложного и практически ориентированного Docker-образа. Разработать многоэтапный Dockerfile для веб-приложения на Spring Boot с подключением к PostgreSQL, реализовать несколько эндпоинтов, включая работу с внешними ресурсами. Опубликовать образ в DockerHub.

ХОД РАБОТЫ

## Задание №1

Создадим простое Ktor приложение, которое будет собираться и развертываться в Docker. Оно будет выводить значения переменных окружения, файлы в томе, а также добавленный во время сборки README.md файл. Исходный код приложения приведен на рисунке 1.

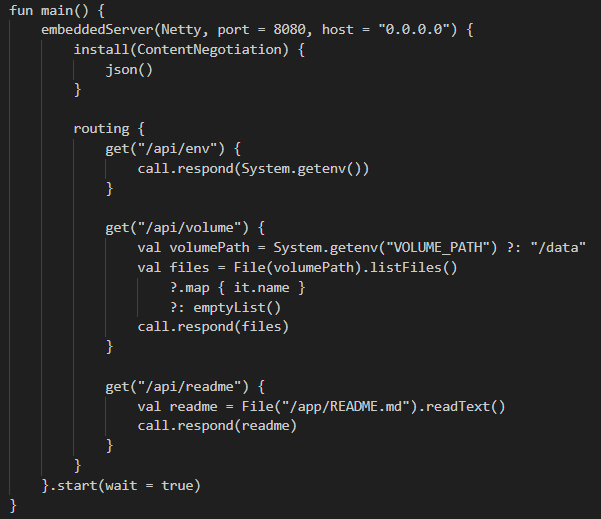


Рисунок 1 – Исходный код приложения

Создадим Dockerfile, в котором опишем все этапы сборки приложения (рисунок 2).

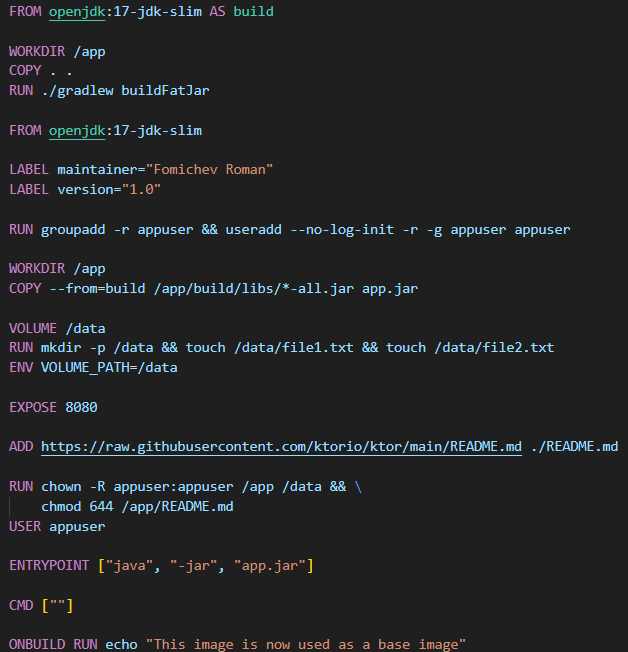
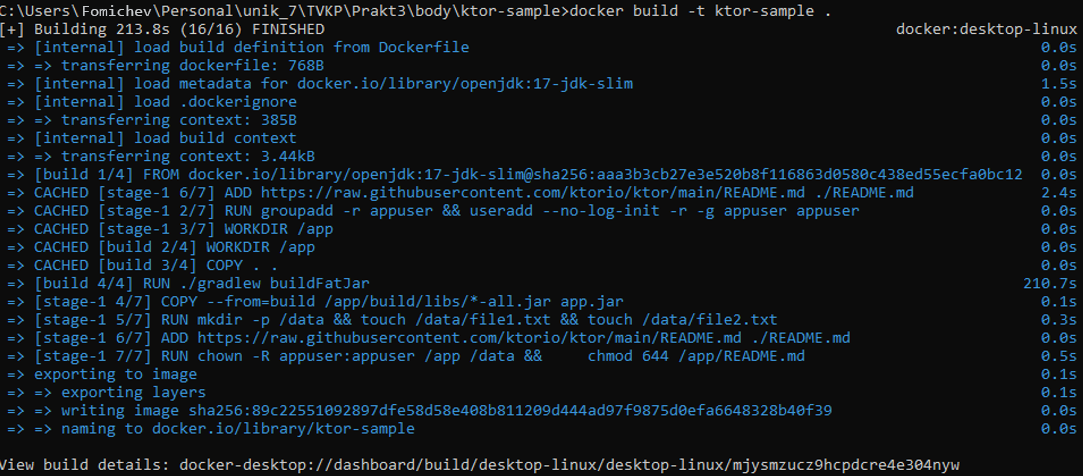


Рисунок 2 – Dockerfile для первого задания

Созданный Dockerfile описывает многоэтапный процесс сборки и настройки Java-приложения:

1. Этап сборки (build):
   * Использует образ openjdk:17-jdk-slim
   * Копирует исходный код и выполняет сборку JAR-файла
2. Основной этап:
   * Базируется на том же образе openjdk:17-jdk-slim
   * Добавляет метки с информацией о мейнтейнере и версии
   * Создает непривилегированного пользователя для запуска приложения
   * Копирует собранный JAR-файл из этапа сборки
   * Настраивает том для данных
   * Устанавливает переменную окружения
   * Открывает порт 8080
   * Загружает README.md файл
   * Настраивает права доступа
   * Устанавливает точку входа для запуска приложения
   * Добавляет команду ONBUILD для выполнения при использовании этого образа как базового

Соберем образ и запустим контейнер на основе собранного образа (рисунок 3).



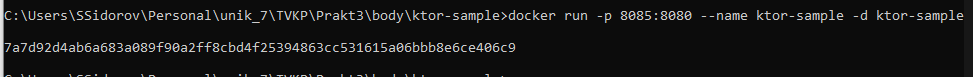
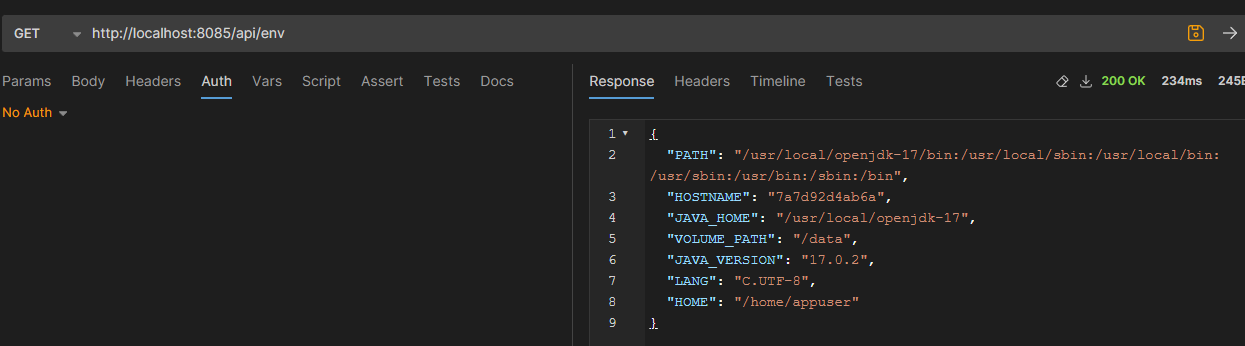
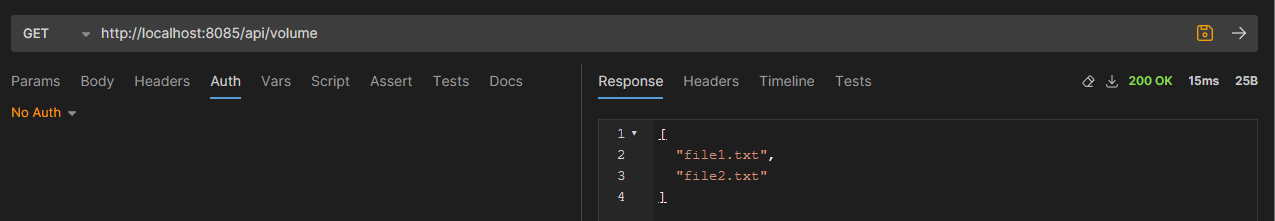


Рисунок 3 – Сборка образа и запуск контейнера с Ktor приложением

Проверим, что все конечные точки доступны (рисунок 4).





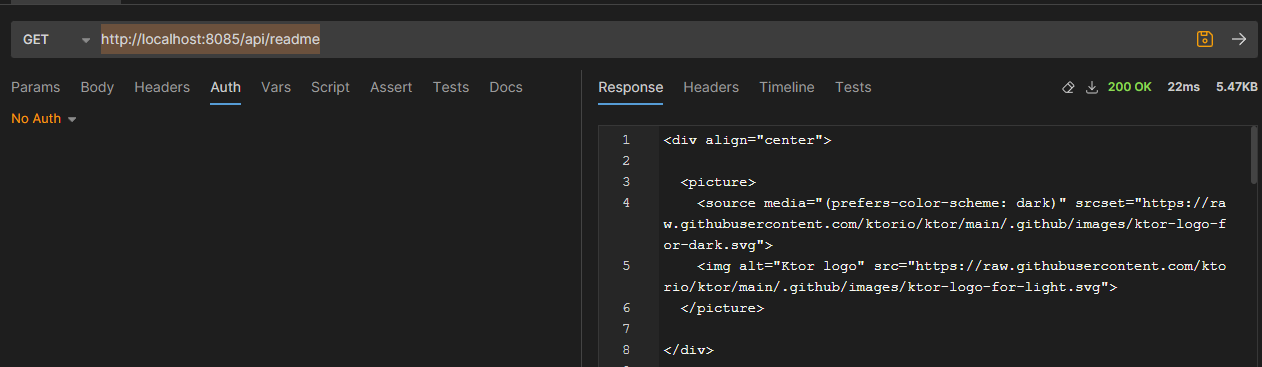


Рисунок 4 – Проверка работы Ktor приложения

## Задание №2

Создадим Spring Boot приложение с требуемой функциональностью. Исходный код приведен на рисунках 5–6.

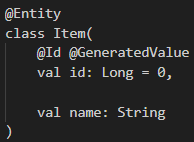


Рисунок 5 – Код модели и репозитория

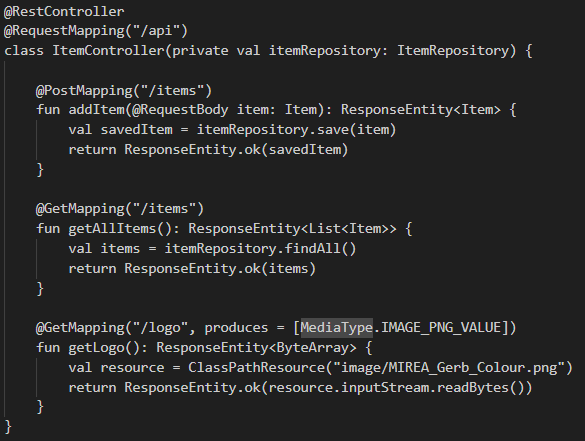


Рисунок 6 – Код контроллера

Конфигурация приложения приведена на рисунке 7.

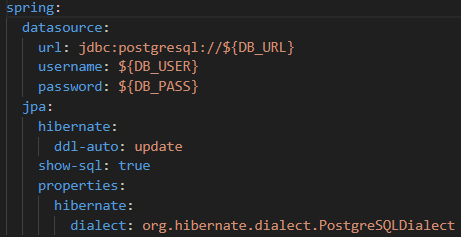


Рисунок 7 – Конфигурация приложения

Создадим Dockerfile для Spring Boot приложения (рисунок 8).

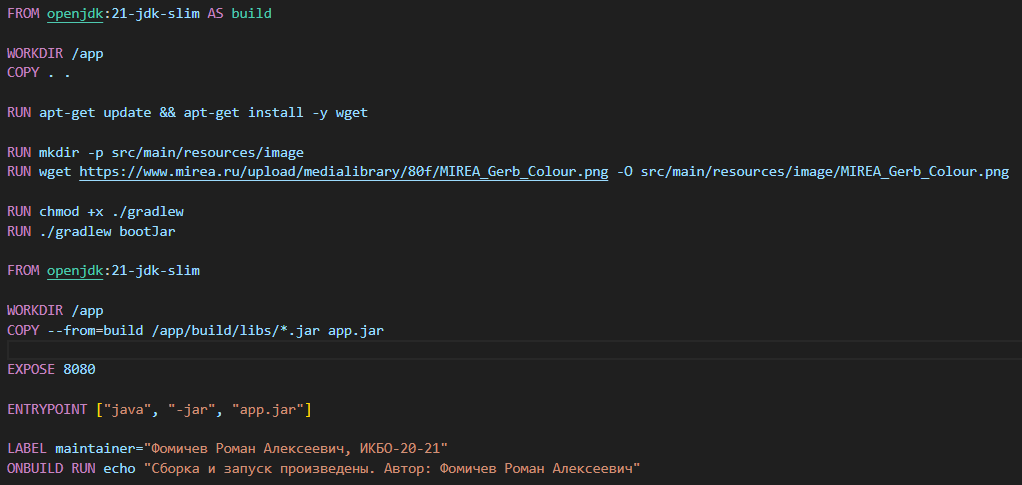


Рисунок 8 – Dockerfile Spring Boot приложения

Данный Dockerfile описывает следующий процесс сборки:

1. Этап сборки (build):
   * Использует Java 21 как базовый образ
   * Устанавливает рабочую директорию и копирует файлы проекта
   * Устанавливает wget для скачивания изображения
   * Загружает герб МИРЭА в ресурсы проекта
   * Собирает JAR-файл с помощью Gradle
2. Этап развертывания:
   * Также использует Java 21, но в минимальной конфигурации
   * Копирует собранный JAR из этапа сборки
   * Открывает порт 8080 для веб-сервиса
   * Задает команду запуска приложения
   * Добавляет метку с информацией об авторе
   * Использует ONBUILD для вывода сообщения при использовании этого образа как базового

Соберем и запустим контейнер с приложением (рисунок 9).

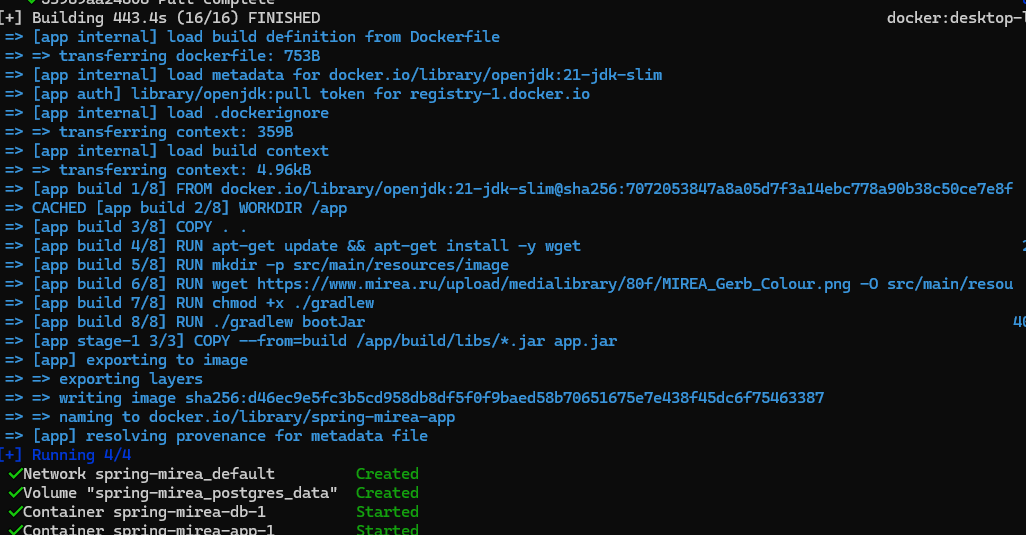
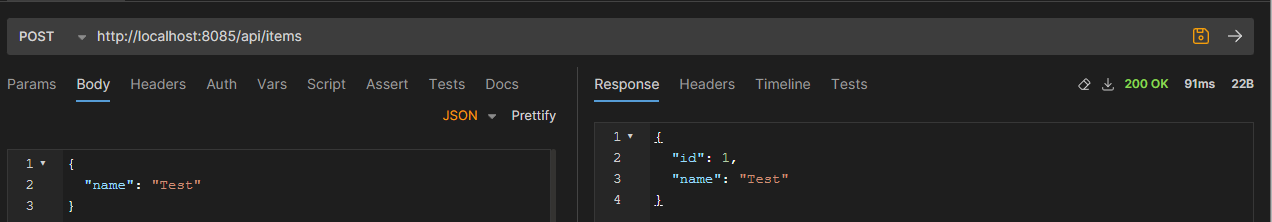
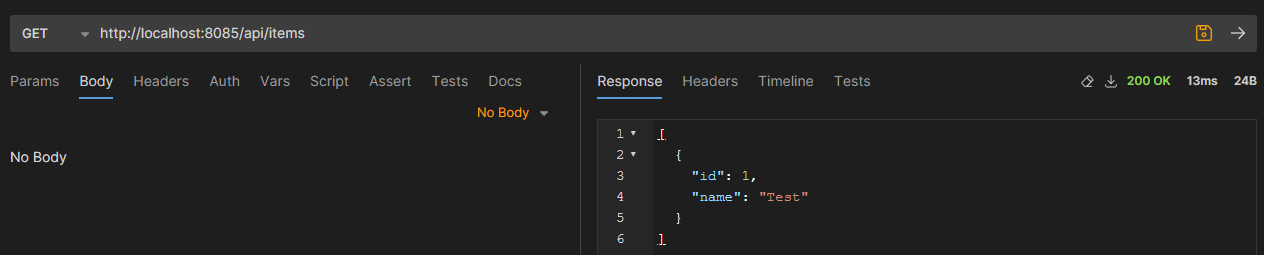


Рисунок 9 – Сборка и запуск Spring Boot приложения

Проверка функционала показана на рисунке 10.





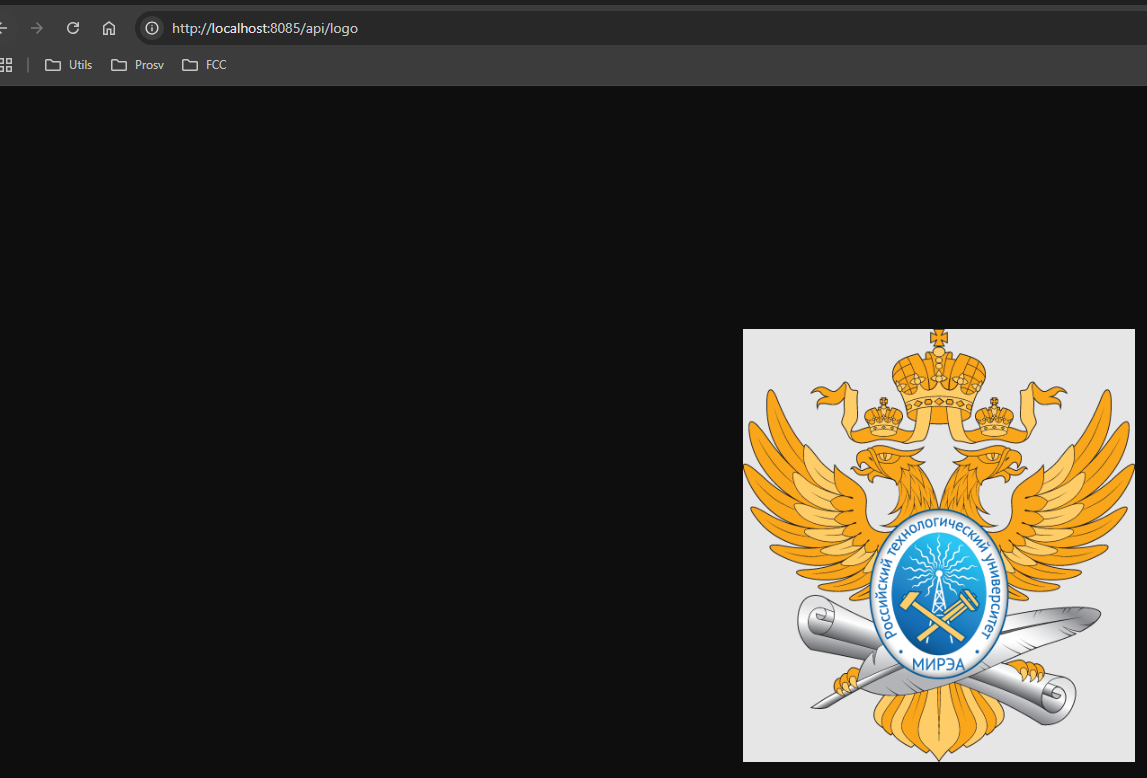
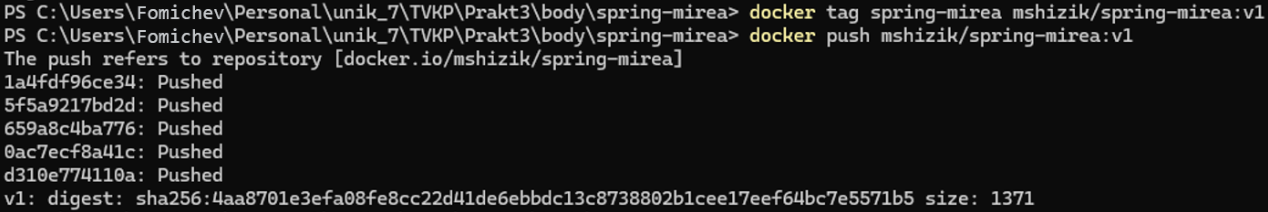


Рисунок 10 – Проверка работоспособности приложения

Теперь загрузим образ на DockerHub (рисунок 11). Результат показан на рисунке 12.

Рисунок 11 – Загрузка образа на DockerHub

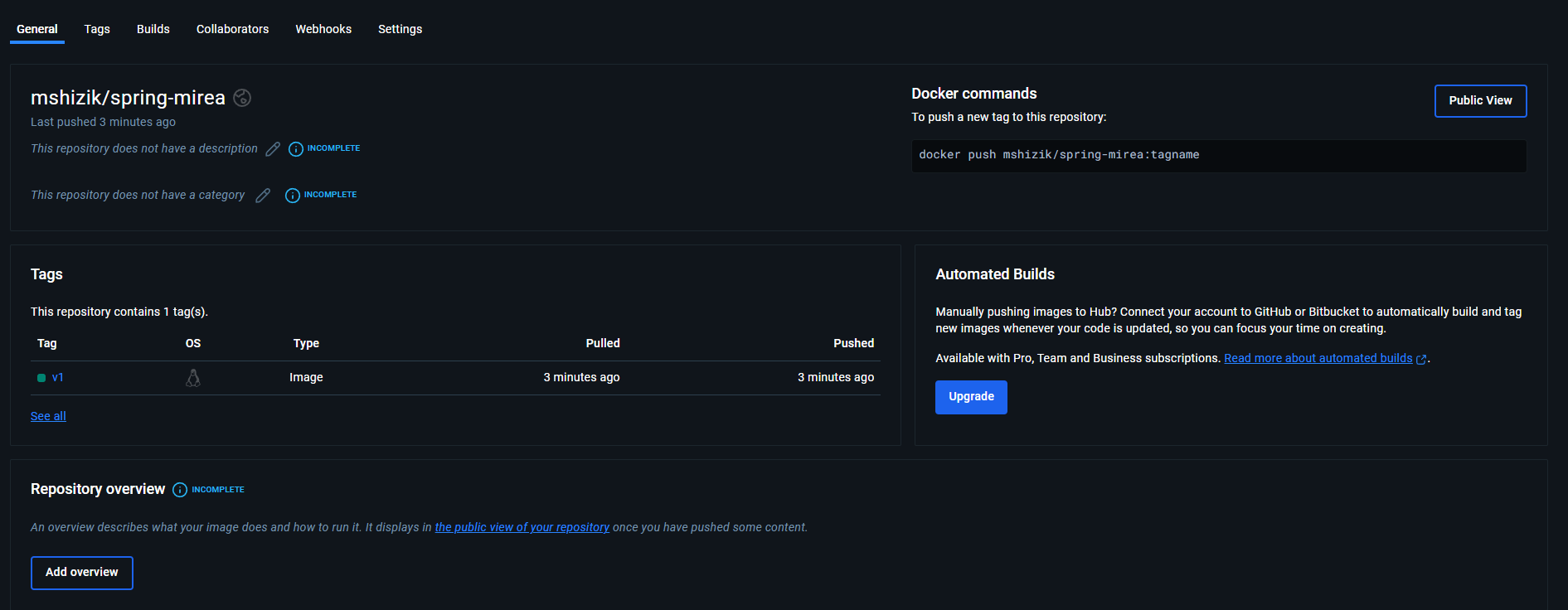


Рисунок 12 – Образ на DockerHub

ВЫВОД

В ходе этой практической работы мы значительно расширили наши знания и навыки в области контейнеризации с использованием Docker. Мы научились создавать эффективные многоэтапные Dockerfile, оптимизированные для Java-приложений на базе Spring Boot.

Особое внимание было уделено интеграции внешних ресурсов в процесс сборки и настройке рабочей среды контейнера. Мы успешно разработали Docker-образ, который не только включает наше веб-приложение, но и учитывает аспекты безопасности и оптимизации размера.

ОТВЕТЫ НА ВОПРОСЫ К ПРАКТИЧЕСКОЙ РАБОТЕ

1. Процесс запуска приложения внутри контейнера Linux с использованием Docker включает следующие шаги:

1.1 Создание Docker-образа на основе Dockerfile

1.2 Запуск контейнера из созданного образа с помощью команды docker run

1.3 Docker демон создает изолированное пространство для контейнера

1.4 Внутри контейнера запускается указанная в Dockerfile команда (ENTRYPOINT или CMD)

1.5 Приложение начинает работу в изолированной среде контейнера

2. Образ Docker — это шаблон, содержащий набор инструкций для создания контейнера. Он включает в себя код приложения, библиотеки, зависимости, инструменты и другие файлы, необходимые для работы приложения. Образы используются для создания и запуска контейнеров, обеспечивая согласованность и портативность приложений.

3. Dockerfile и Docker-Compose соотносятся следующим образом:

Dockerfile описывает, как построить отдельный Docker-образ

Docker-Compose — это инструмент для определения и запуска многоконтейнерных приложений

Docker-Compose использует образы, созданные на основе Dockerfile, для определения сервисов в многоконтейнерном приложении

4. Dockerfile - это текстовый файл, содержащий набор инструкций для автоматического создания Docker-образа. Он определяет базовый образ, копирует файлы, устанавливает зависимости, настраивает среду выполнения и задает команды для запуска приложения внутри контейнера.

5. Политики перезапуска контейнера в Docker:

5.1 no: не перезапускать автоматически (по умолчанию)

5.2 on-failure[:max-retries]: перезапуск при сбое, с опциональным ограничением числа попыток

5.3 always: всегда перезапускать контейнер, если он останавливается

5.4 unless-stopped: всегда перезапускать, кроме случаев, когда контейнер был остановлен вручную

6. Возможные состояния контейнеров:

6.1 Created: контейнер создан, но не запущен

6.2 Running: контейнер запущен и работает

6.3 Paused: контейнер приостановлен

6.4 Stopped: контейнер остановлен

6.5 Exited: контейнер завершил работу

6.6 Dead: контейнер в нерабочем состоянии из-за ошибки или сбоя

6.7 Restarting: контейнер в процессе перезапуска

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 50 вопросов по Docker, которые задают на собеседованиях, и ответы на них | Хабр. — Текст: электронный [сайт]. — URL: https://habr.com/ru/company/southbridge/blog/528206

2. Docker Documentation | Docker Documentation — Текст: электронный [сайт]. — URL: https://docs.docker.com/

3. Что такое режим Docker Swarm и когда его использовать? — CloudSavvy ИТ | Cpab. — Текст: электронный [сайт]. — URL: https://cpab.ru/chto-takoe-rezhim-docker-swarm-i-kogda-ego-ispo4lzovat-cloudsavvy-it/

4. Dockerfile reference | Docker Documentation — Текст: электронный [сайт]. — URL: https://docs.docker.com/engine/reference/builder/