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

ПРАКТИЧЕСКИЕ РАБОТЫ

ПО МДК 04.02 «Обеспечение качества функционирования компьютерных систем»

|  |  |
| --- | --- |
| Выполнил студент  группы П50-3-22  Пушкин Илья Александрович | Проверил преподаватель  \_\_\_\_\_\_\_\_\_\_\_ К.С. Образцова  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 года |

Москва 2025

[ПРАКТИЧЕСКАЯ РАБОТА №1 3](#_Toc208865660)

[ПРАКТИЧЕСКАЯ РАБОТА №2 12](#_Toc208865661)

# ПРАКТИЧЕСКАЯ РАБОТА №1

Цель работы: Выполнить сборки и развертывание контейнеров из примера, и описать своими словами то, что именно вы сделали и как. Описать вкладки в Docker (включая вкладки внутри собранного образа и контейнера) Написать свои Doсkerfile и собрать контейнеры с программами из архива. Залить готовую практическую ( все программы + Dockerfile к каждой из них) на Gitlab или Github и прикрепить к заданию ссылку.

Ход работы:

1. Dart программа

Для дарта используется FROM dart:stable, далее прописать LABEL version = "1.0.1". который является необязательной частью, затем WORKDIR /dartPR1 для создания рабочей директории внутри контейнера, далее COPY . /dartPR1/ для переноса файлов из текущей директории внутрь контейнера, и CMD ["dart","calc.dart"] позволяет запустить файл calc.dart с помощью dart соответственно.

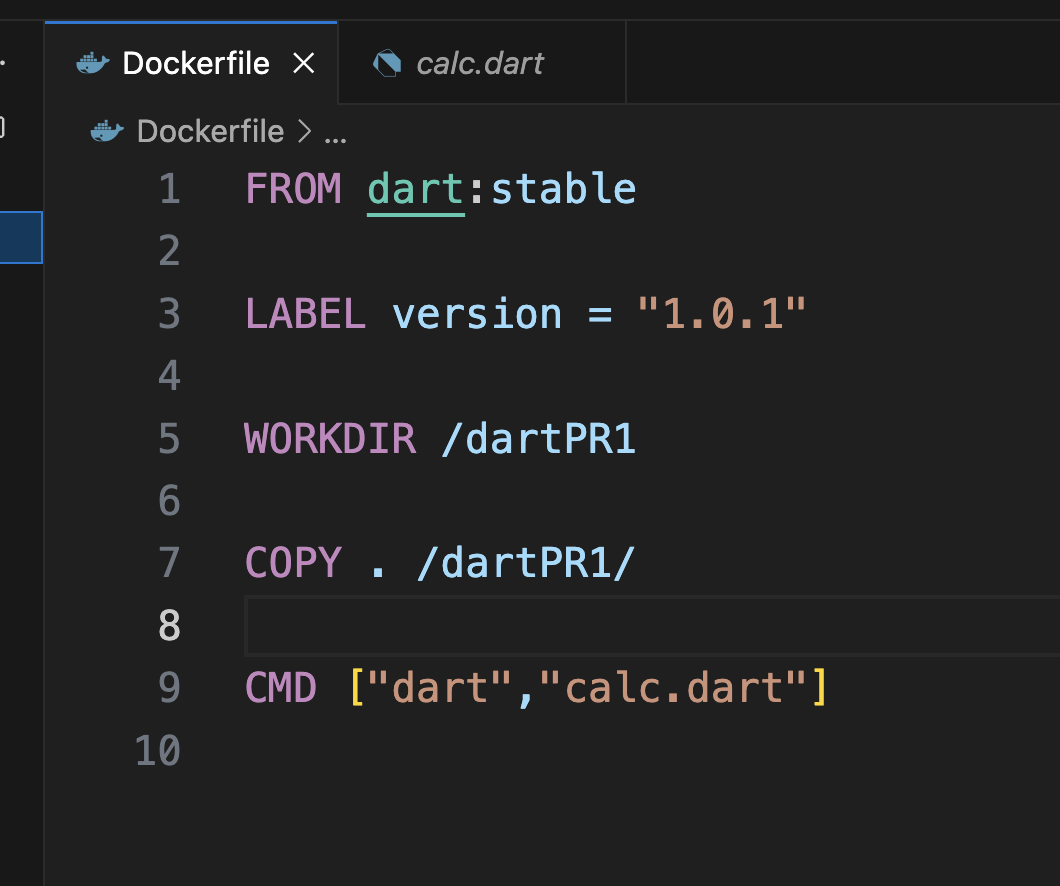


Рисунок 1 – Итоговый Dockerfile к Dart программе

Далее, необходимо собрать образ и запустить контейнер. Сборка образа происходит по команде docker build -t dartpr1-my . флаг -t используется для задания имени образа, это не обязательно, но тогда название будет ужасным. Точка указывает на текущую директорию (там где искать все файлы, которые мы дальше копируем). Затем необходимо запустить контейнер при помощи команды docker run -it --name dokerpr-1-5 dartpri-my. -it создает как бы виртуальный терминал, чтобы дальше с этим можно было работать, как с обычным терминалом в консоли. --name dokerpr-1-5 задаем имя контейнеру и далее имя образа из которого создается контейнер.

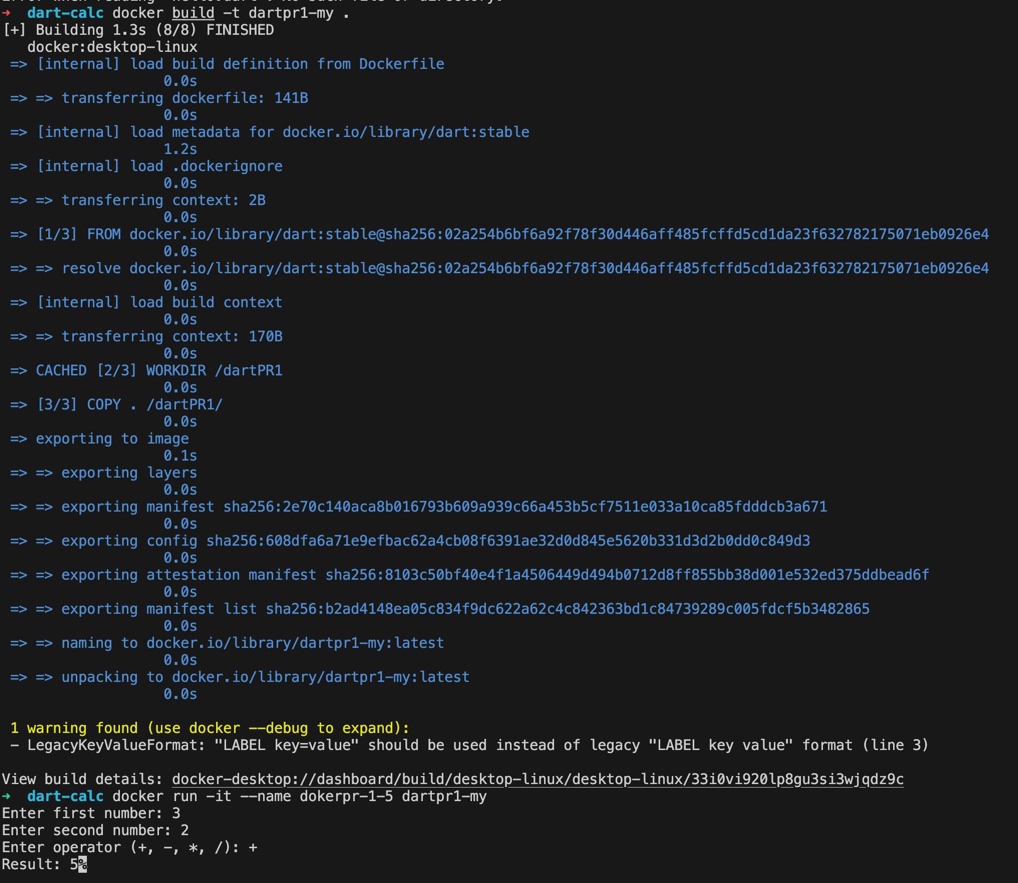


Рисунок 2 – Сборка образа и запуск его контейнера Dart

1. Python калькулятор

Для питона используется FROM python:alpine, далее прописать LABEL version = "1.0.1", затем WORKDIR /pyyPR1 для создания рабочей директории внутри контейнера, далее COPY . /pyyPR1/ для переноса файлов из текущей директории внутрь контейнера, и CMD [ "python" , "calc.py" ] позволяет запустить файл calc.py с помощью python соответственно.

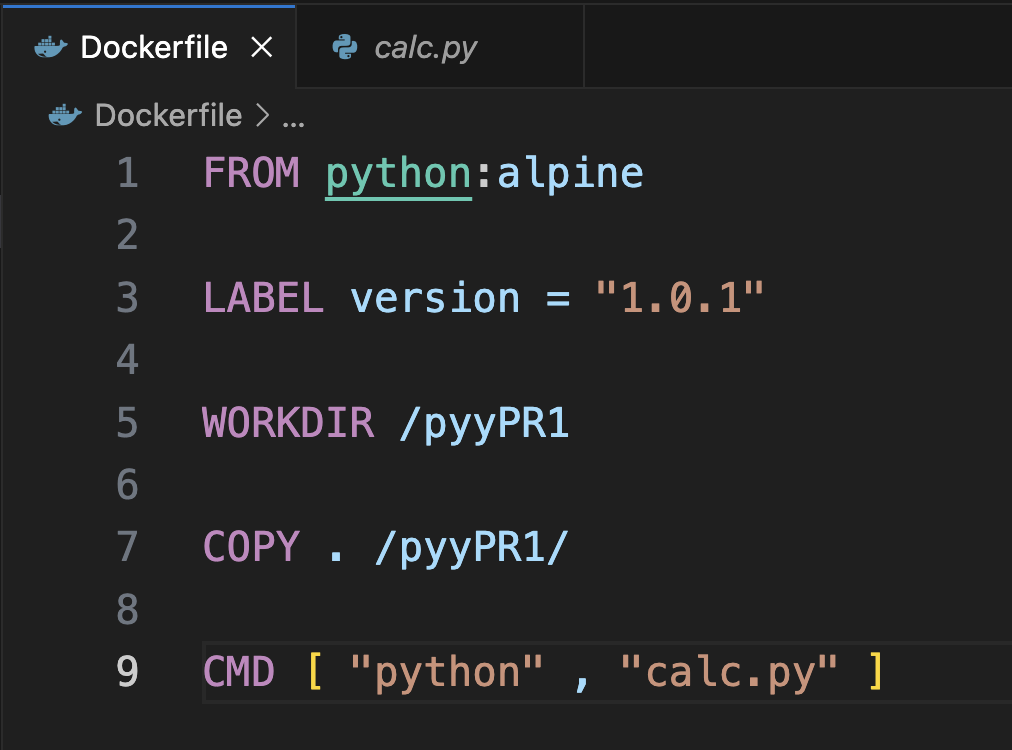


Рисунок 3 - Итоговый Dockerfile к Python-калькулятор

Далее, необходимо собрать образ и запустить контейнер точно так же как и в предыдущих пунктах.

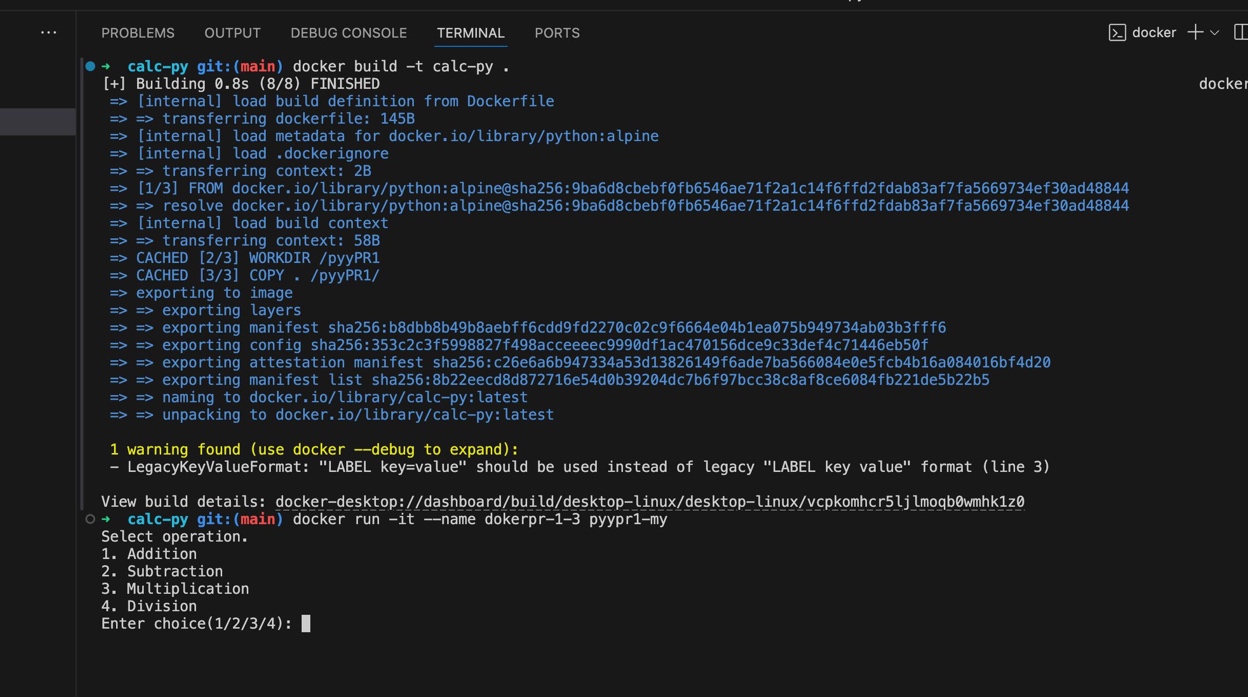


Рисунок 4 - Сборка образа и запуск его контейнера Python-калькулятор

1. Html

Для вебки испоьзуется FROM nginx:alpine. Используем nginx, так как сайты, далее копируем файлы в контейнер COPY . /usr/share/nginx/html и затем указываем порт внутри контейнера EXPOSE 80.

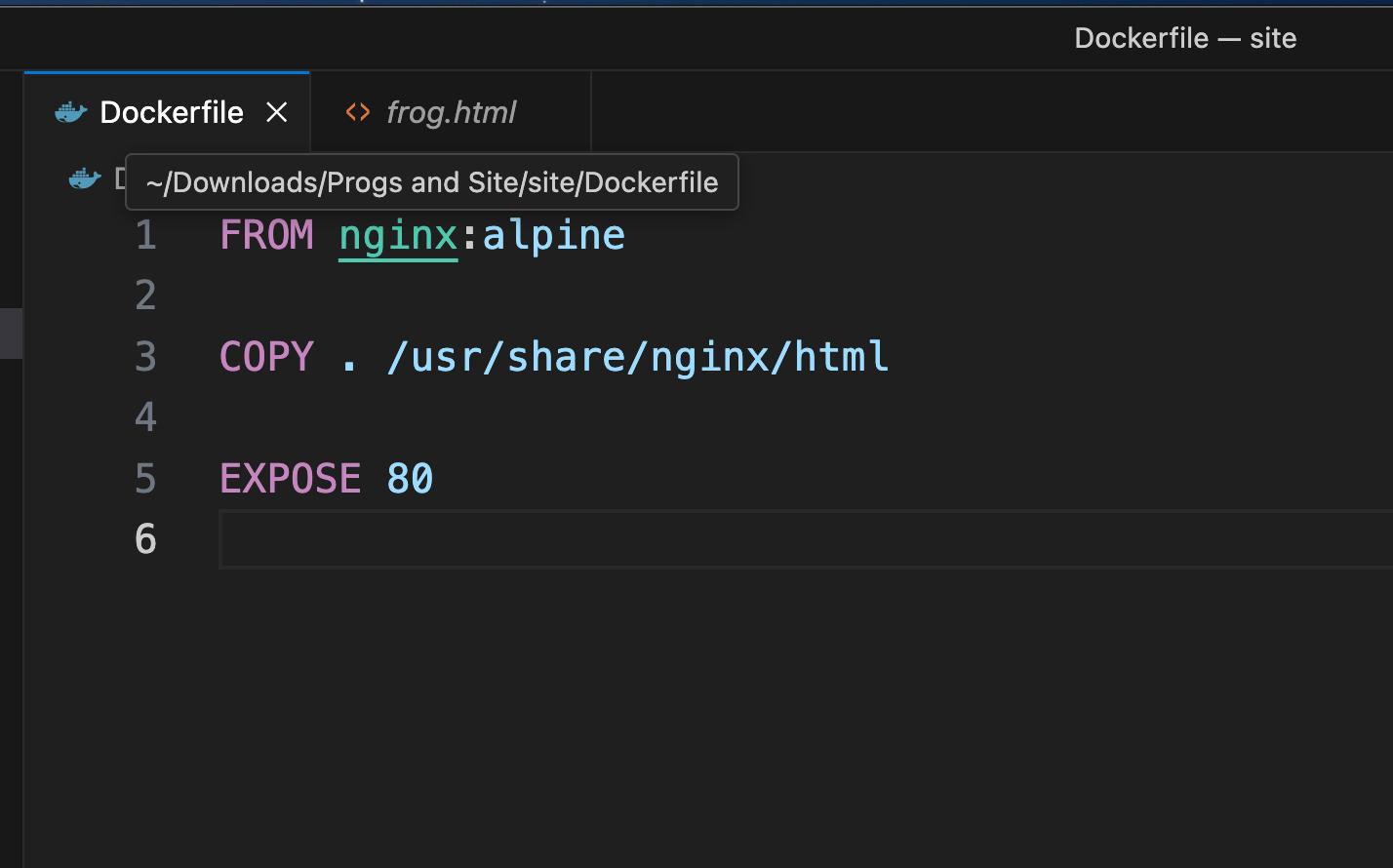


Рисунок 5 - Итоговый Dockerfile к WEB

Далее, необходимо собрать образ и запустить контейнер. Сборка образа происходит по команде docker build -t dartpr1-my . флаг -t используется для задания имени образа, это не обязательно, но тогда название будет ужасным. Точка указывает на текущую директорию (там где искать все файлы, которые мы дальше копируем). Затем необходимо запустить контейнер при помощи команды docker run -d -p 8080:80 --name site-pr1-container site-pr1. С помощью -d запускаем контейнер в фоне, -p 8080:80 подбрасываем порты (порт 80 мы указали в EXPOSE). О остальном все то же самое.

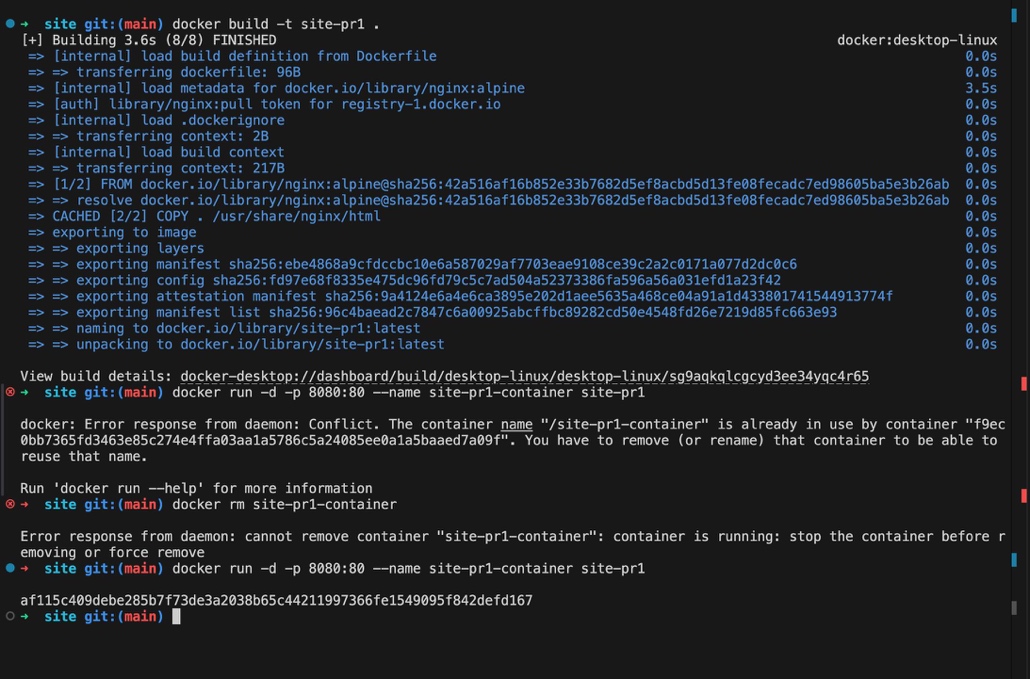


Рисунок 6 - Сборка образа и запуск его контейнера WEB

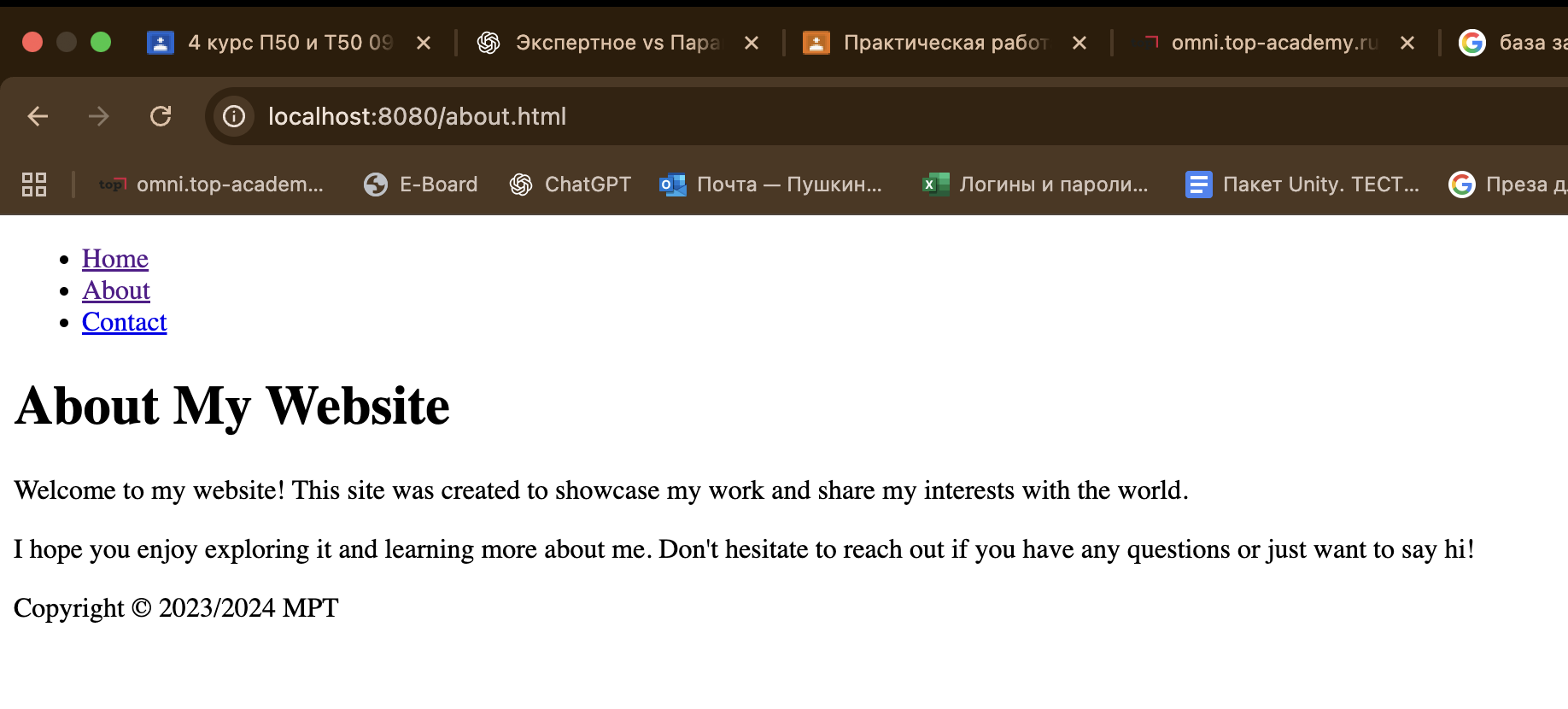


Рисунок 7 – Результат запуска WEB контейнера.

1. Java калькулятор

Для джавы используется FROM openjdk:alpine, далее прописываем WORKDIR /javaaPR1 и COPY . /javaaPR1/ для создания директории и для переноса в нее файлов проекта. В джаве, необходимо прописать RUN javac calc.java так как в файле с калькулятором есть класс, докеру надо указать, что у нас есть класс, иначе, он его не найдет. И теперь прописываем CMD ["java", "calc"] для того, чтобы запустить класс calc в файле с джавой, который мы указали выше в строчке run, с помощью java.

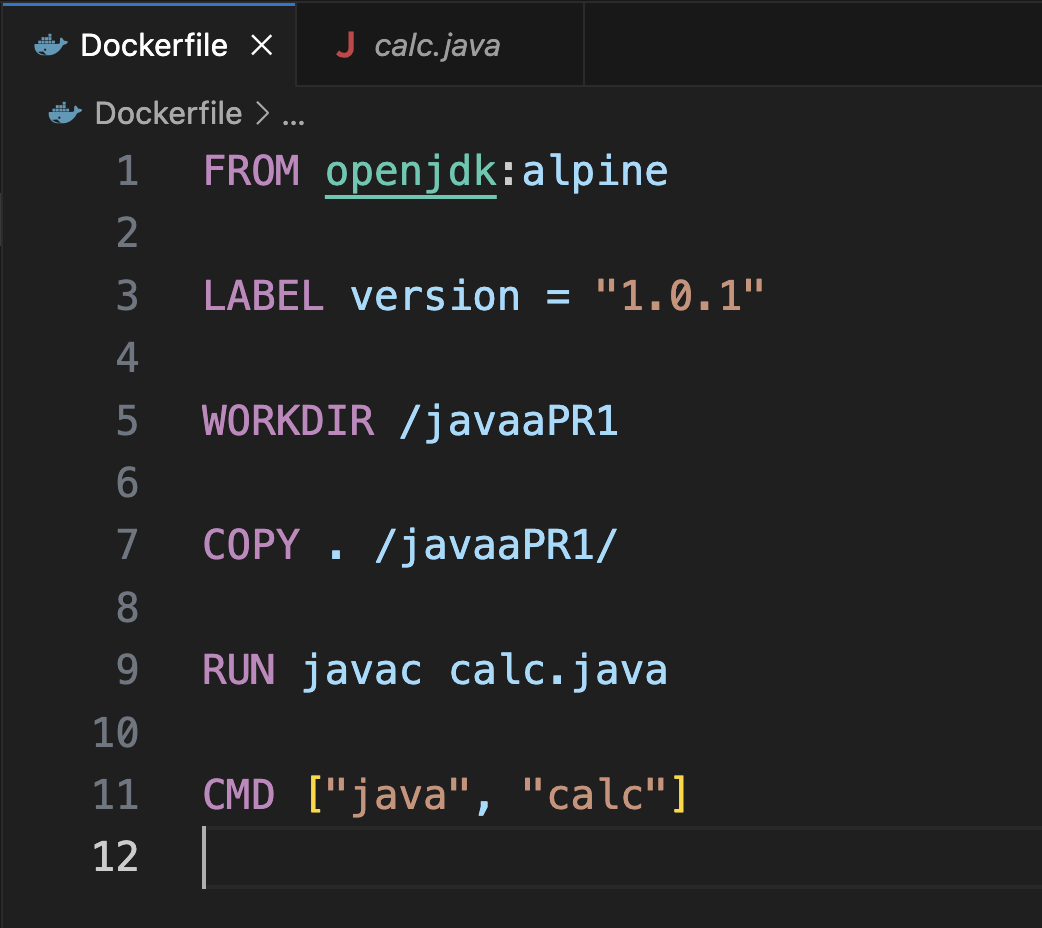


Рисунок 8 - Итоговый Dockerfile к Java-калькулятору

Далее, необходимо собрать образ и запустить контейнер точно так же, как и в предыдущих пунктах.

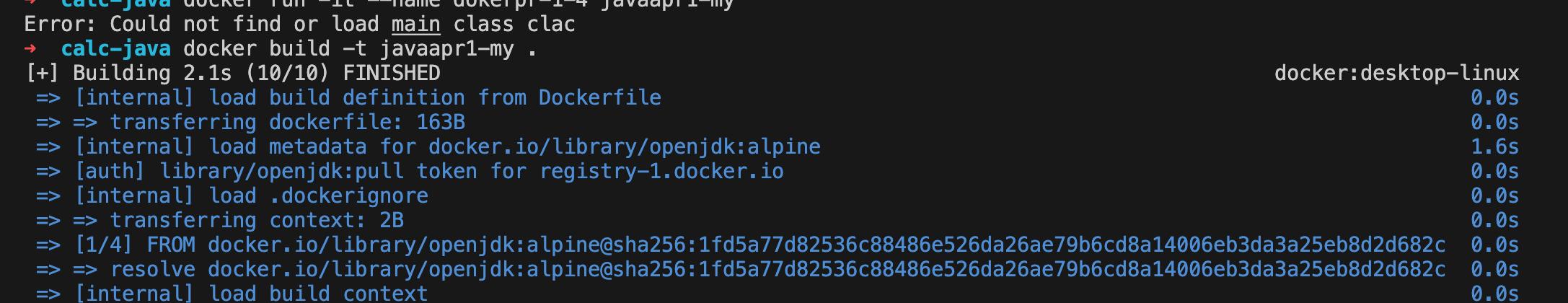


Рисунок 9 - Сборка образа Java-калькулятора

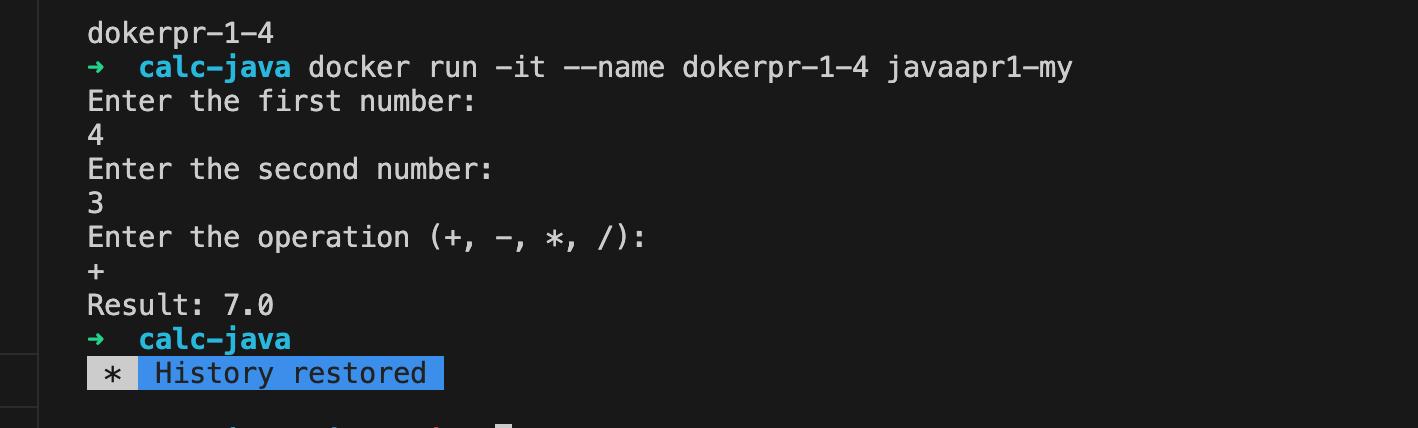


Рисунок 10 - Запуск контейнера Java-калькулятора

1. Java игра

Здесь все тоже самое, что и в калькуляторе, различие в создаваемой директории, так как не могут быть одинаковые WORKDIR /javaPR1 и COPY . /javaPR1/. И запуск игры по тому же сценарию RUN javac game.java и CMD [ "java" , "game" ].

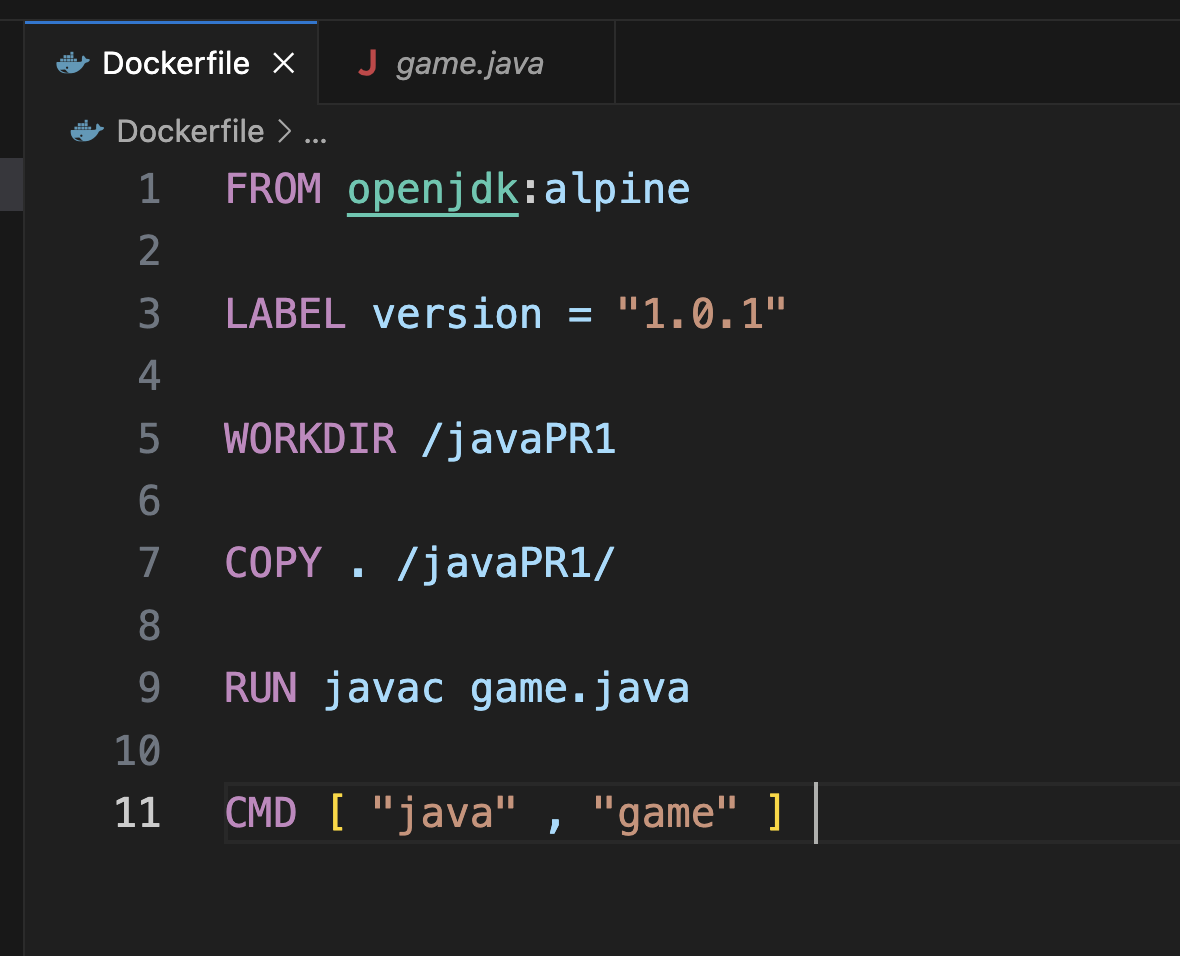


Рисунок 11 - Итоговый Dockerfile к Java-игре

Далее, необходимо собрать образ и запустить контейнер точно так же, как и в предыдущих пунктах.

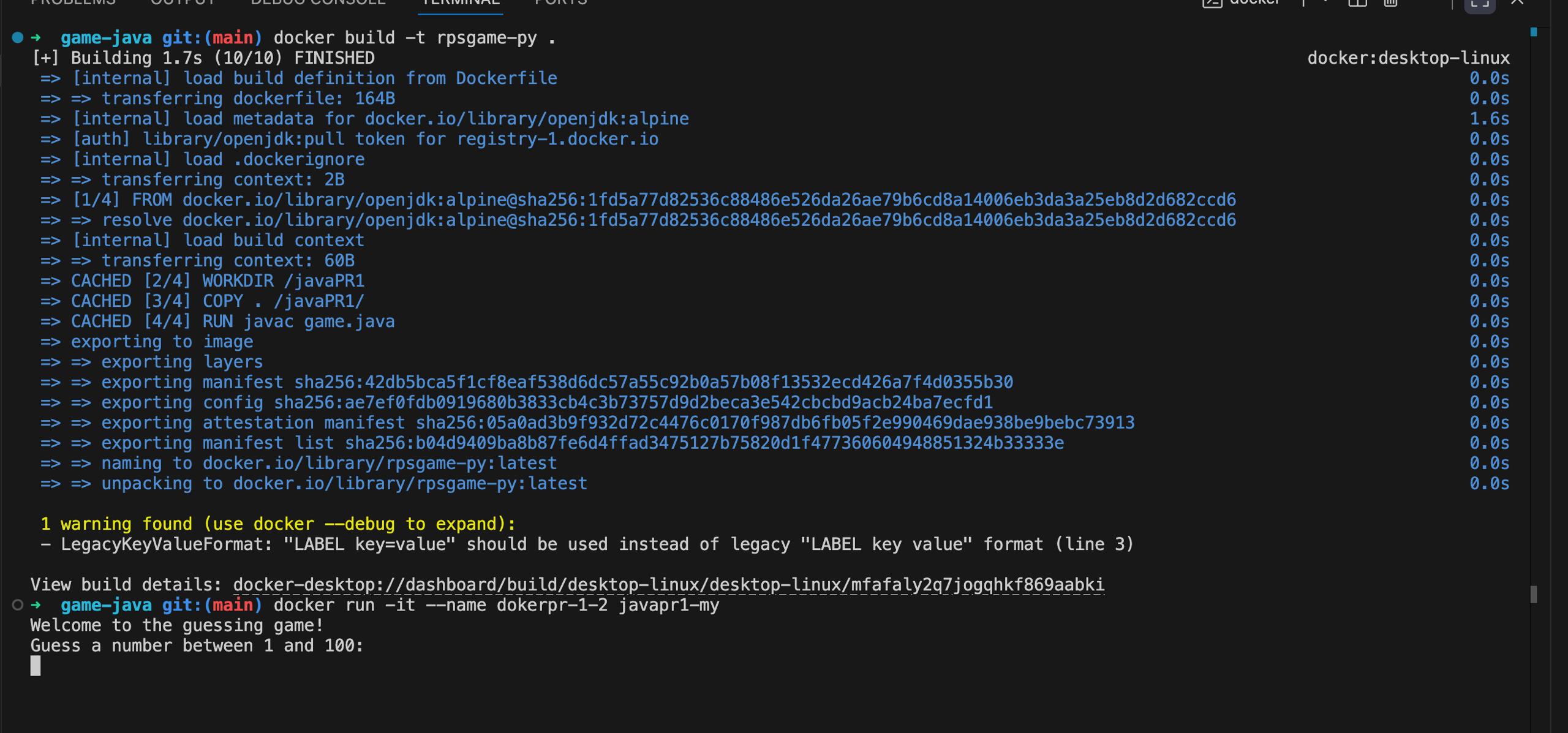


Рисунок 12 - Сборка образа и запуск его контейнера Java-игры

1. Python-игра

Здесь все то же самое, что и в Python-калькуляторе. Отличаются только директории и файлы которые мы запускаем.

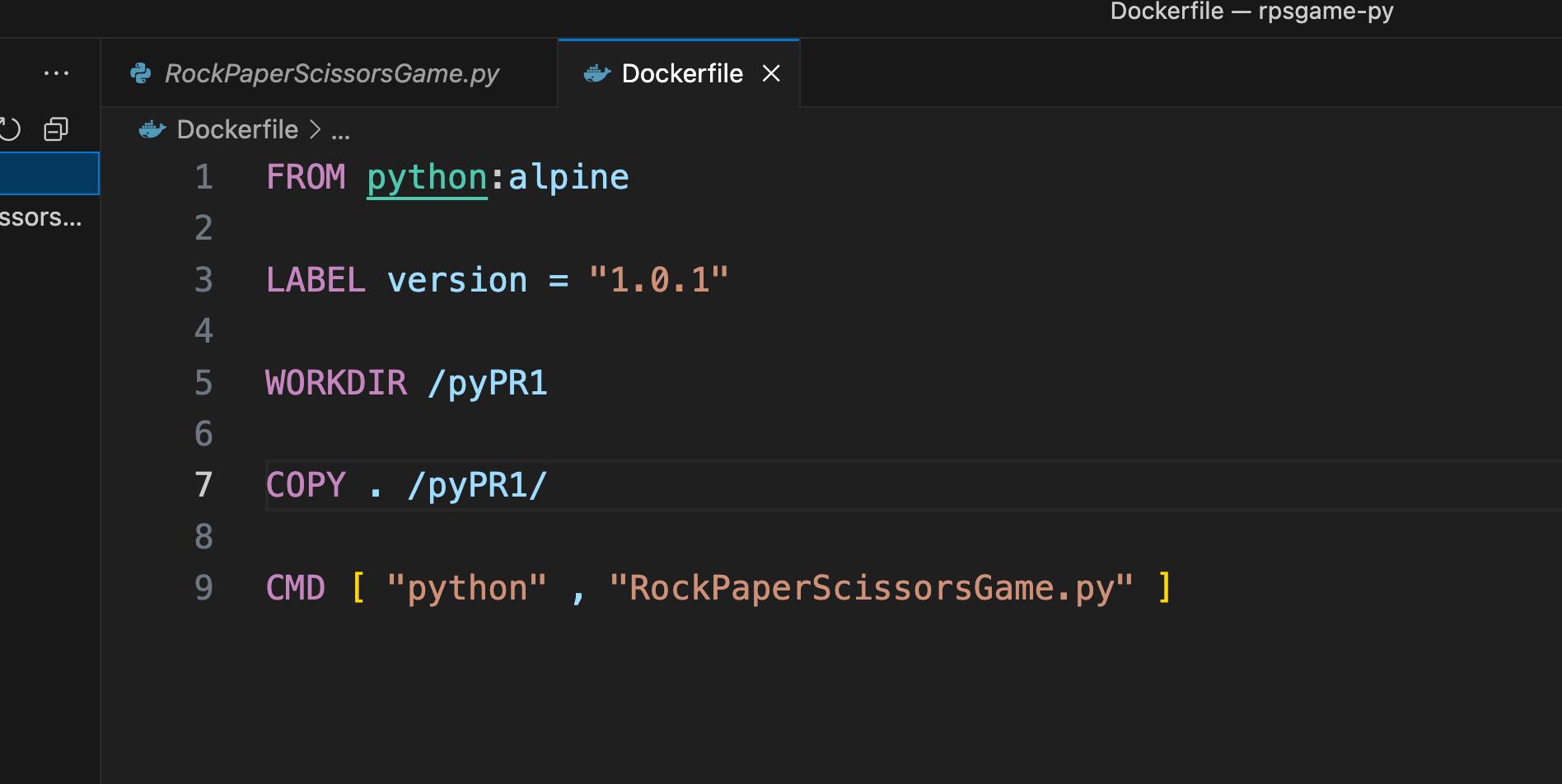


Рисунок 13 - Итоговый Dockerfile к Python-игре

Далее, необходимо собрать образ и запустить контейнер точно так же, как и в предыдущих пунктах.

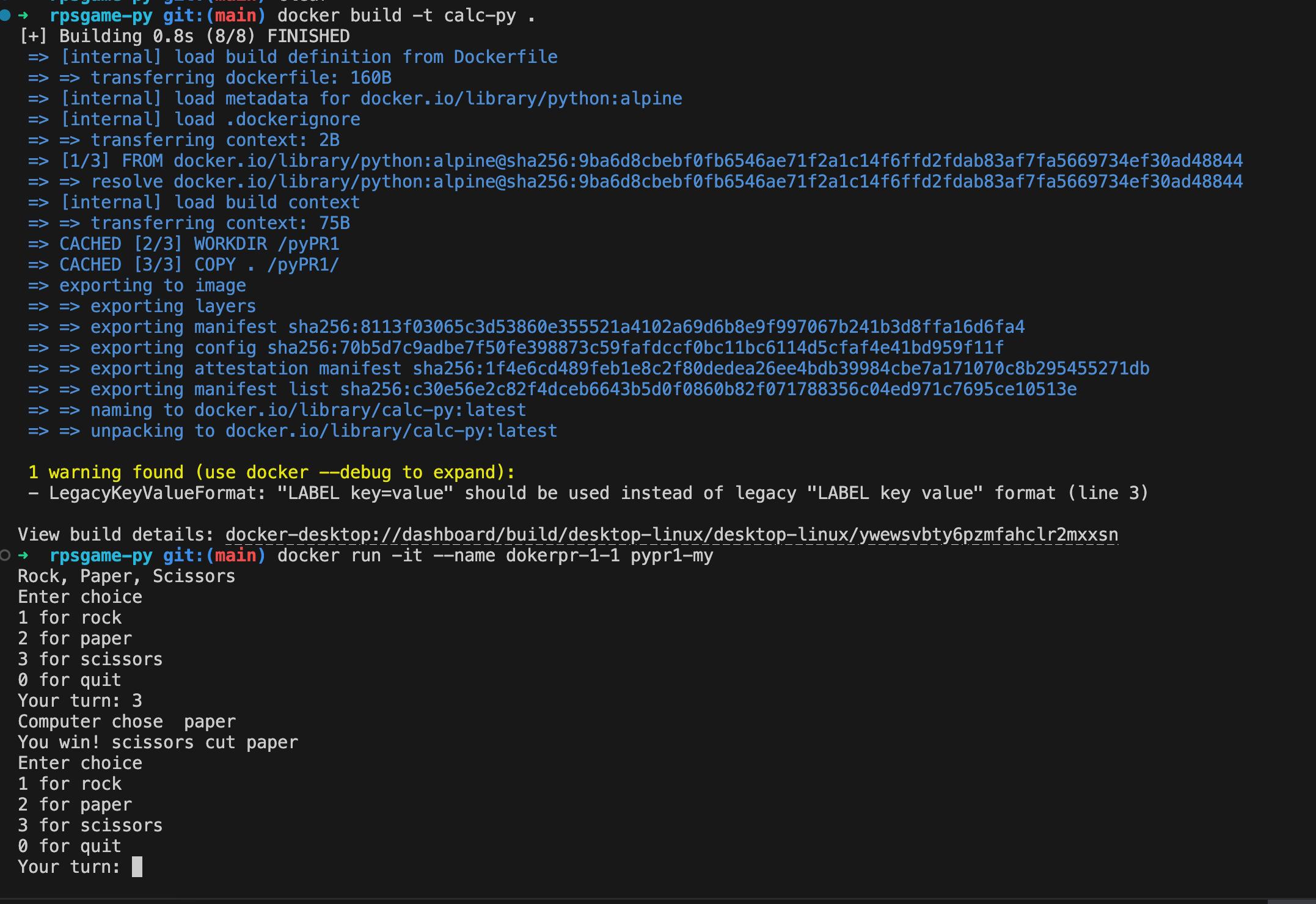


Рисунок 14 - Сборка образа и запуск его контейнера Python-игры

7. Описание вкладок Docker Desktop.

Containers - Здесь список контейнеров — запущенных и остановленных.

Images - Это список всех образов.

Volumes - Здесь отображаются тома

Builds - Это раздел про сборку образов.

Logs – Здесь показываются логи работы контейнера (что он выводит в консоль).

Inspect – Здесь можно посмотреть подробную «техническую информацию» о контейнере (IP, порты, настройки и т. д.).

Bind mounts – Здесь отображаются папки на компьютере, которые подключены внутрь контейнера.

Exec – Здесь можно запустить команду внутри контейнера (как будто ты зашёл в него через терминал).

Files – Здесь видна файловая система контейнера (все папки и файлы внутри него).

Stats – Здесь показываются ресурсы, которые потребляет контейнер (CPU, память, сеть, диск).

Info – Здесь показывается общая информация о сборке образа (название, идентификатор, ошибки).

Source – Здесь можно посмотреть, из каких файлов/директорий собирался образ.

Logs – Здесь отображаются логи процесса сборки (что делал Docker при docker build).

History – Здесь видна история слоёв образа (какие команды из Dockerfile создавали текущий результат).

Вывод: В ходе практической работы было выполнено развертывание контейнеров по примерам, изучены и описаны вкладки Docker Desktop и их назначение. Были написаны собственные Dockerfile, собраны контейнеры с программами из архива и загружены в репозиторий GitLab/GitHub. Работа позволила закрепить навыки по сборке и запуску контейнеров, а также работе с Dockerfile.

# ПРАКТИЧЕСКАЯ РАБОТА №2

Цель работы: Развернуть приложение time-db из примера в Docker. Описать каждый пункт -почему именно эти команды и компоненты вы создаете.

Залить готовую практическую на Gitlab или Github и прикрепить к заданию ссылку.

Ход работы:

1. Создать в обоих компонентах Dockerfile.

Эти Dockerfile никак не отличаются от предыдущей практической работы.

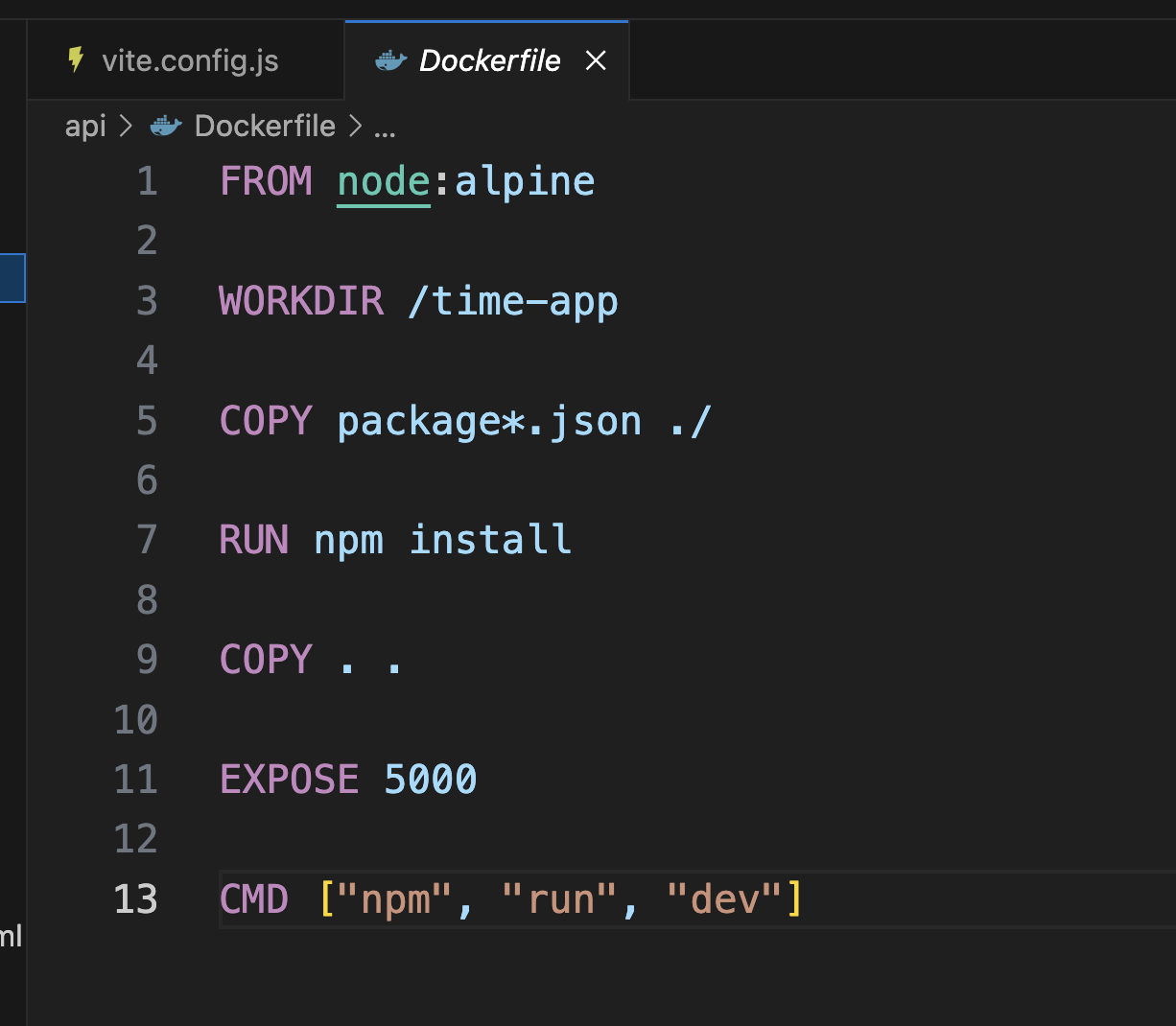


Рисунок 15 - Dockerfile api

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 16 - Dockerfile frontend

1. Создать docker-compose.yaml.

Расширение файла yaml будет означать словарь данных. То есть словарь настроек docker compose, который будет запускаться в этом решении.

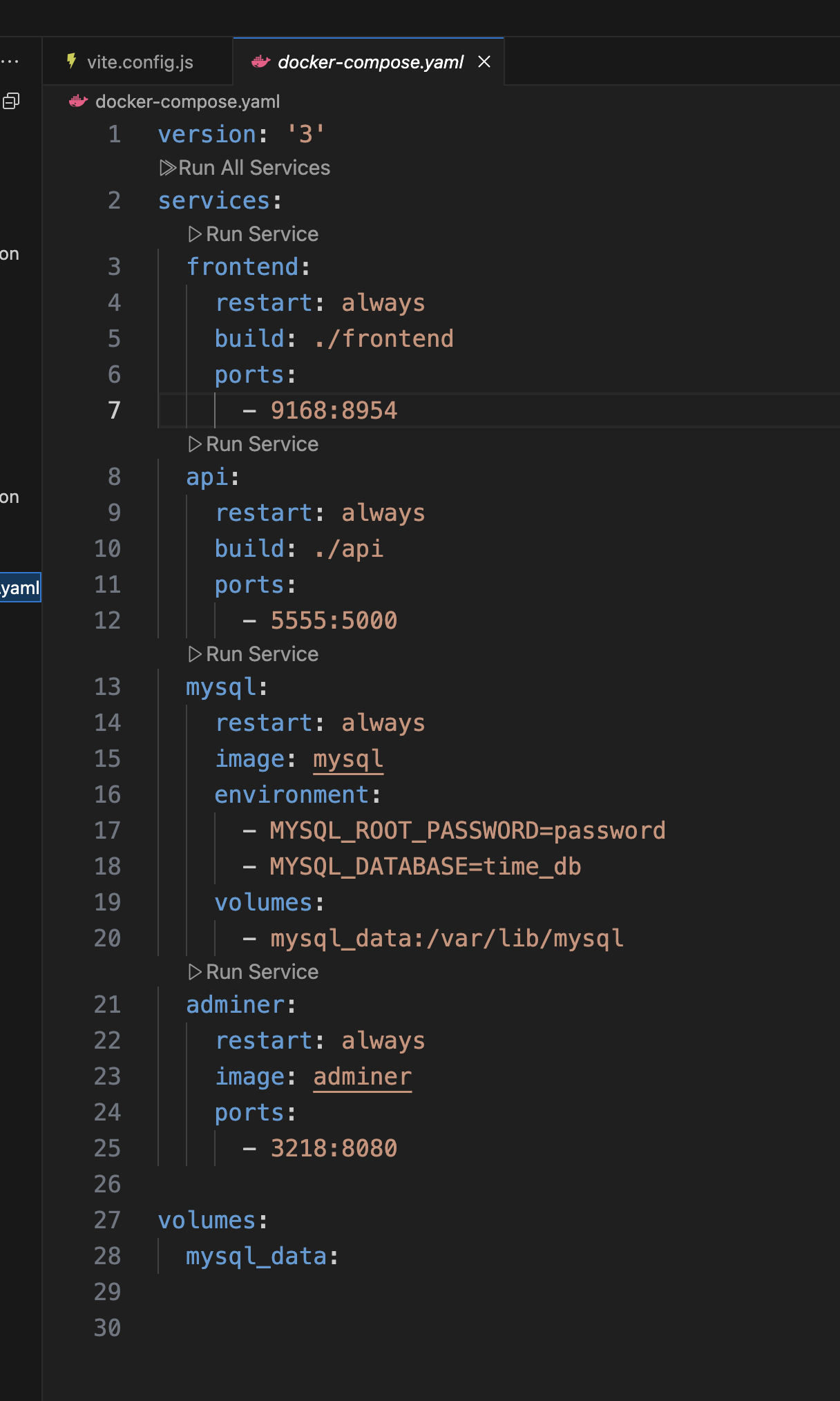


Рисунок 17 - docker-compose.yaml

В файле docker-compose в начале указывается version: '3', это версия синтаксиса, которая определяет какие возможности доступны. Далее идёт блок services, где описываются все контейнеры. Первый сервис называется frontend, для него задана команда restart: always, что означает автоматический перезапуск при сбое или перезагрузке. Директива build: ./frontend указывает что образ будет собран из Dockerfile, расположенного в папке frontend. В секции ports указано 9168:8954, это проброс портов, где 9168 доступен на хосте, а 8954 внутри контейнера.

Следующий сервис api устроен аналогично. У него тоже стоит restart: always для автоматического перезапуска. build: ./api означает что образ соберётся из Dockerfile в папке api. Проброс портов 5555:5000 позволяет обращаться к сервису на порту 5555 снаружи, а внутри контейнера он работает на 5000.

Далее идёт сервис mysql, который используется как база данных. У него задано restart: always, что гарантирует перезапуск. В качестве образа используется официальный mysql. В блоке environment заданы переменные окружения: MYSQL\_ROOT\_PASSWORD=password задаёт пароль для пользователя root, а MYSQL\_DATABASE=time\_db автоматически создаёт базу данных с именем time\_db. В блоке volumes подключается том mysql\_data к пути /var/lib/mysql внутри контейнера, чтобы данные базы сохранялись при перезапуске.

Следующий сервис adminer нужен для удобного управления базой данных через веб-интерфейс. У него тоже стоит restart: always. В качестве образа используется официальный adminer. Проброс портов 3218:8080 означает что доступ к веб-интерфейсу будет по адресу localhost:3218, а внутри контейнера он слушает на порту 8080.

В конце файла идёт раздел volumes, где объявлен именованный том mysql\_data. Он используется в сервисе mysql для хранения данных и обеспечивает их сохранность между перезапусками контейнеров.

1. Запуск docker compose

Команда docker-compose build --no-cache запускает процесс сборки всех сервисов, описанных в файле docker-compose. build означает, что будут собираться образы на основе Dockerfile, указанных в каждом сервисе. Ключ --no-cache используется для того, чтобы сборка происходила полностью с нуля, без использования ранее сохранённых слоёв кэша. Это гарантирует, что все изменения в коде или зависимостях будут учтены, и контейнеры получат абсолютно свежие образы.

Команда docker-compose up запускает все сервисы, описанные в файле docker-compose, и создает на их основе контейнеры. up объединяет процесс создания и запуска, то есть если образы ещё не были собраны, то они будут собраны автоматически.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 18 - Запуск docker-compose

1. Результат выполнения работы

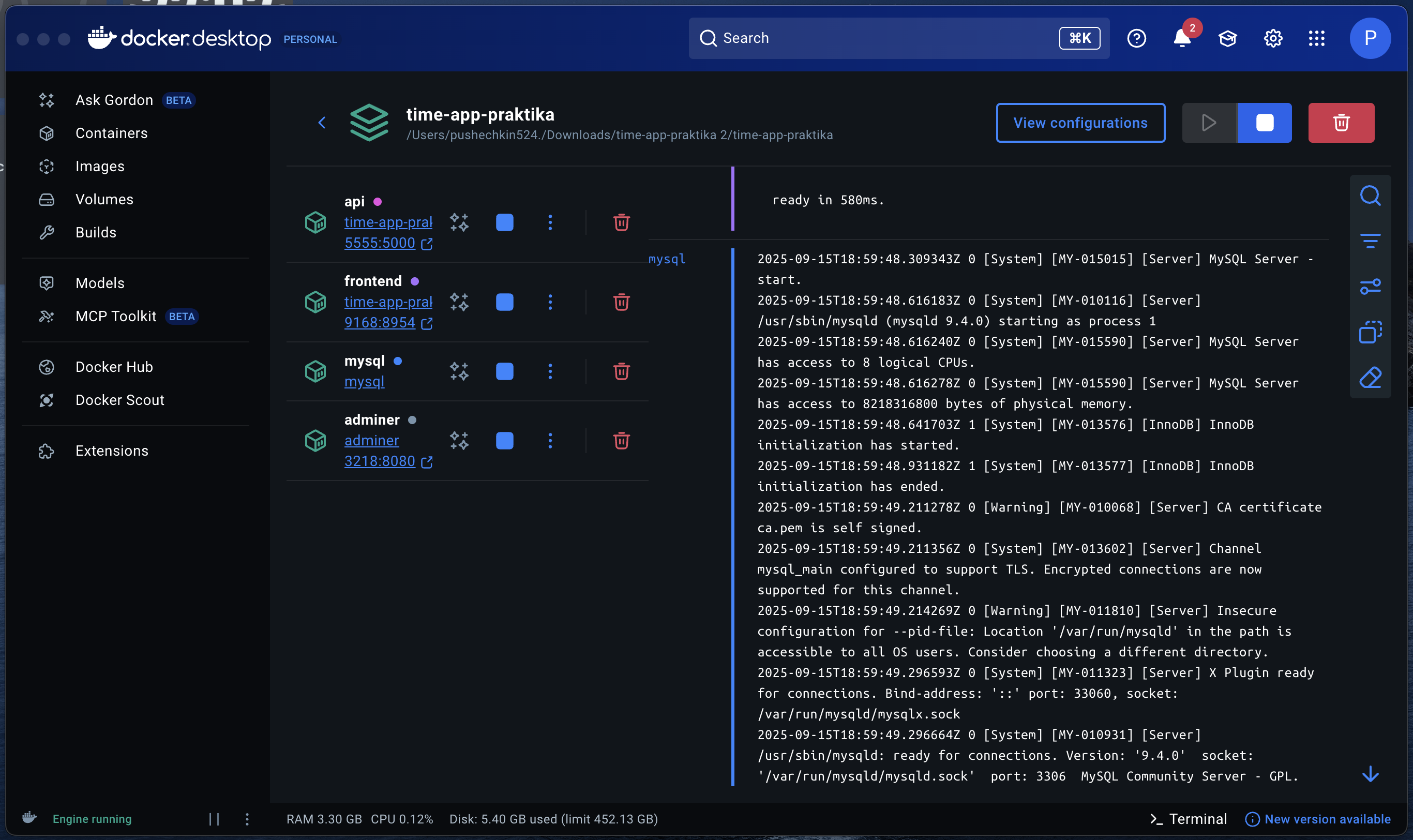


Рисунок 19 - Все компоненты в Docker desktop

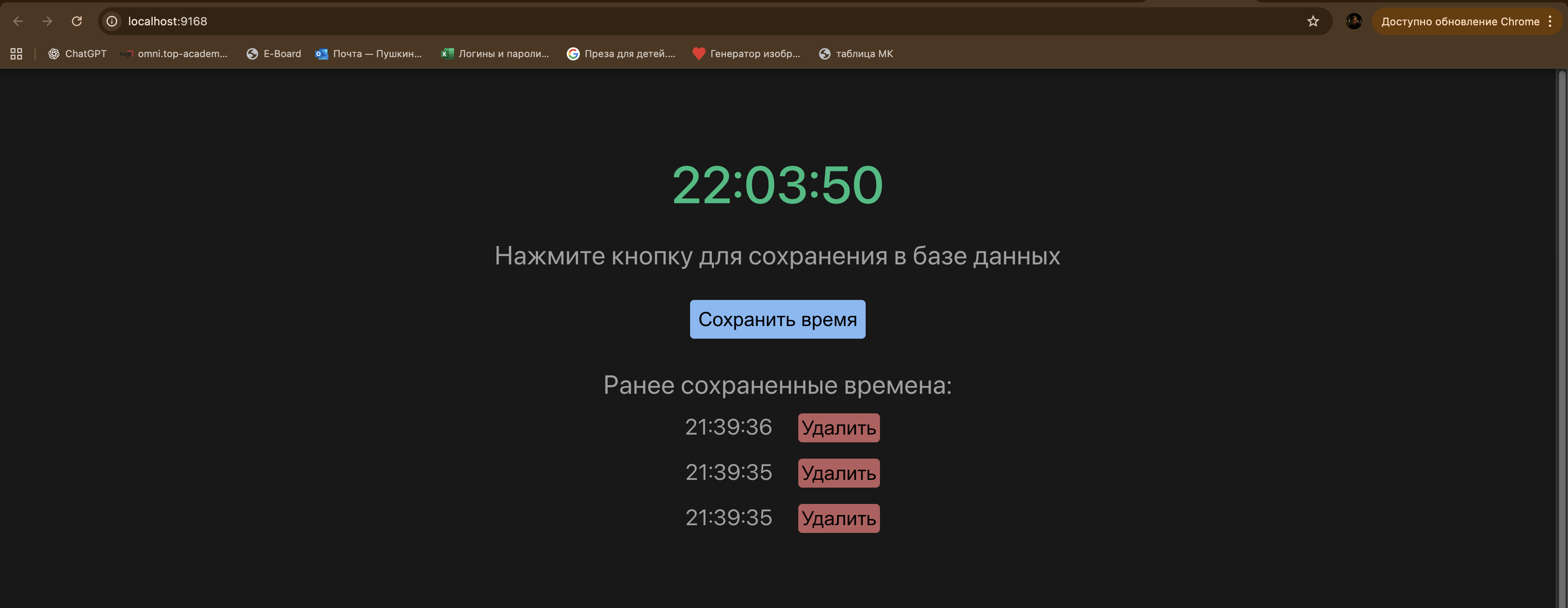


Рисунок 20 - frontend

Изображение выглядит как программное обеспечение, Мультимедийное программное обеспечение, текст, Графическое программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 21 - adminer

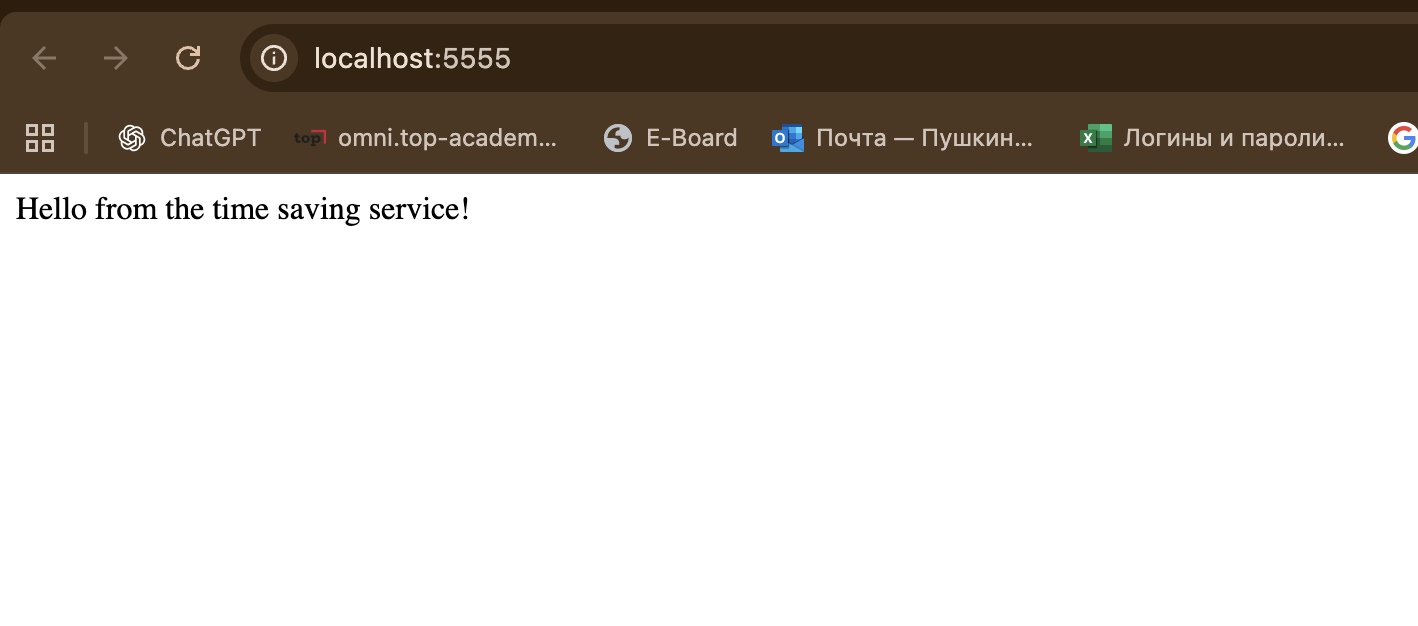


Рисунок 22 - api

Вывод: В ходе практической работы было выполнено развертывание приложения time-db в Docker с использованием docker-compose. Каждый компонент и команда были подробно описаны и обоснованы: почему выбирается именно такой сервис, зачем используется проброс портов, создание томов и переменные окружения. На практике были изучены и закреплены навыки настройки нескольких связанных контейнеров — frontend, api, mysql и adminer, а также их взаимодействия внутри одной сети. Дополнительно рассмотрены команды docker-compose build --no-cache и docker-compose up, их назначение и необходимость при развертывании проекта.