Департамент образования и науки города Москвы Государственное автономное образовательное учреждение высшего образования города Москвы

«Московский городской педагогический университет» Институт цифрового образования

Департамент информатики управления и технологий

Ситьков Александр Вячеславович БД-241м

**Практическая работа 3-1. Dockerfile**

Направление подготовки/специальность

38.04.05 - Бизнес-информатика Бизнес-аналитика и большие данные

(очная форма обучения) Вариант 22

Москва 2024

[Введение 3](#_bookmark0)

[Основная часть 3](#_bookmark1)

[Заключение 1](#_bookmark2)7

Введение

Цель работы: получить практические навыки в написании Dockerfile, научиться

различать и применять хорошие и плохие практики при создании Docker-образов.

Задачи

1. Изучить основные принципы написания Dockerfile.

2. Проанализировать плохие и хорошие практики.

3. Создать два варианта Dockerfile с демонстрацией противоположных подходов.

4. Провести сравнительный анализ результатов.

# Основная часть

**1. Написание хорошего и плохого Dockerfile.**

Для начала, необходимо на виртуальной машине создать 2 директории “dockerfile\_good” и “dockerfile\_bad”.

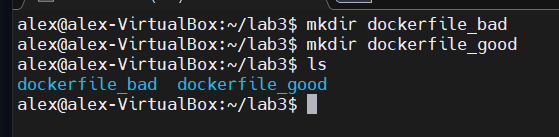


Рисунок 1.1 – 2 директории созданы

Внутри каждой из этих директорий создадим dockerfile.

Содержание плохого dockerfile

from python:latest

workdir /app

copy ./requirements.txt /app/server/requirements.txt copy ./alembic.ini /app/alembic.ini

copy ./src /app/server/src

run pip install --no-cache-dir --upgrade -r /app/server/requirements.txt

cmd ["uvicorn", "server.src.main:app", "--host", "0.0.0.0", "--port", "8000"]

volume /app/data

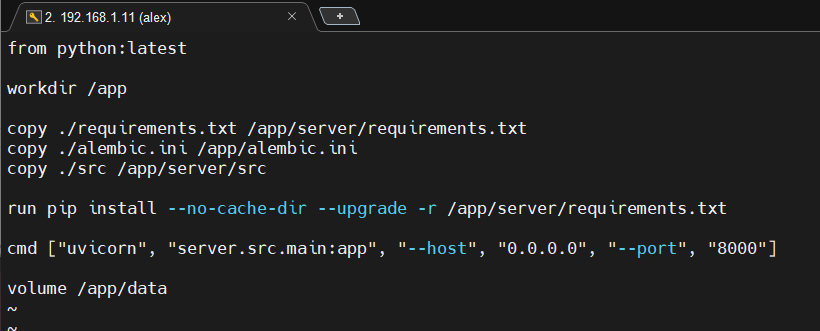


Рисунок - 1.2 Содержимое плохого dockerfile

Содержание хорошего dockerfile

FROM python:3.10

WORKDIR /app

COPY ./requirements.txt /app/server/requirements.txt COPY ./alembic.ini /app/alembic.ini

COPY ./src /app/server/src

RUN apt-get update && pip install --no-cache-dir --upgrade -r

/app/server/requirements.txt

CMD ["uvicorn", "server.src.main:app", "--host", "0.0.0.0", "--port", "8000"] VOLUME /app/data

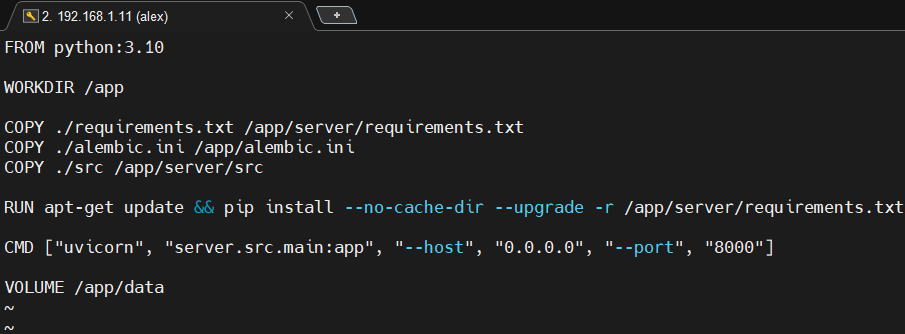


Рисунок 1.3 - Содержимое хорошего dockerfile

Далее из репозитория “[https://github.com/BosenkoTM/DevOps-](https://github.com/BosenkoTM/DevOps-Technologies/tree/main/Lesson%203%20Containerizing%20Applications/lab3_1#%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA)

[Technologies/tree/main/Lesson%203%20Containerizing%20Applications/lab3\_1#%D0%](https://github.com/BosenkoTM/DevOps-Technologies/tree/main/Lesson%203%20Containerizing%20Applications/lab3_1#%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA) [B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA](https://github.com/BosenkoTM/DevOps-Technologies/tree/main/Lesson%203%20Containerizing%20Applications/lab3_1#%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA)”

Перенесем содержимое в обе директории и внутрь их перенесем и заменим докер файлы на вышеописанные.

Далее, начнем построение Docker-образа.

**docker build -t test\_bad .**

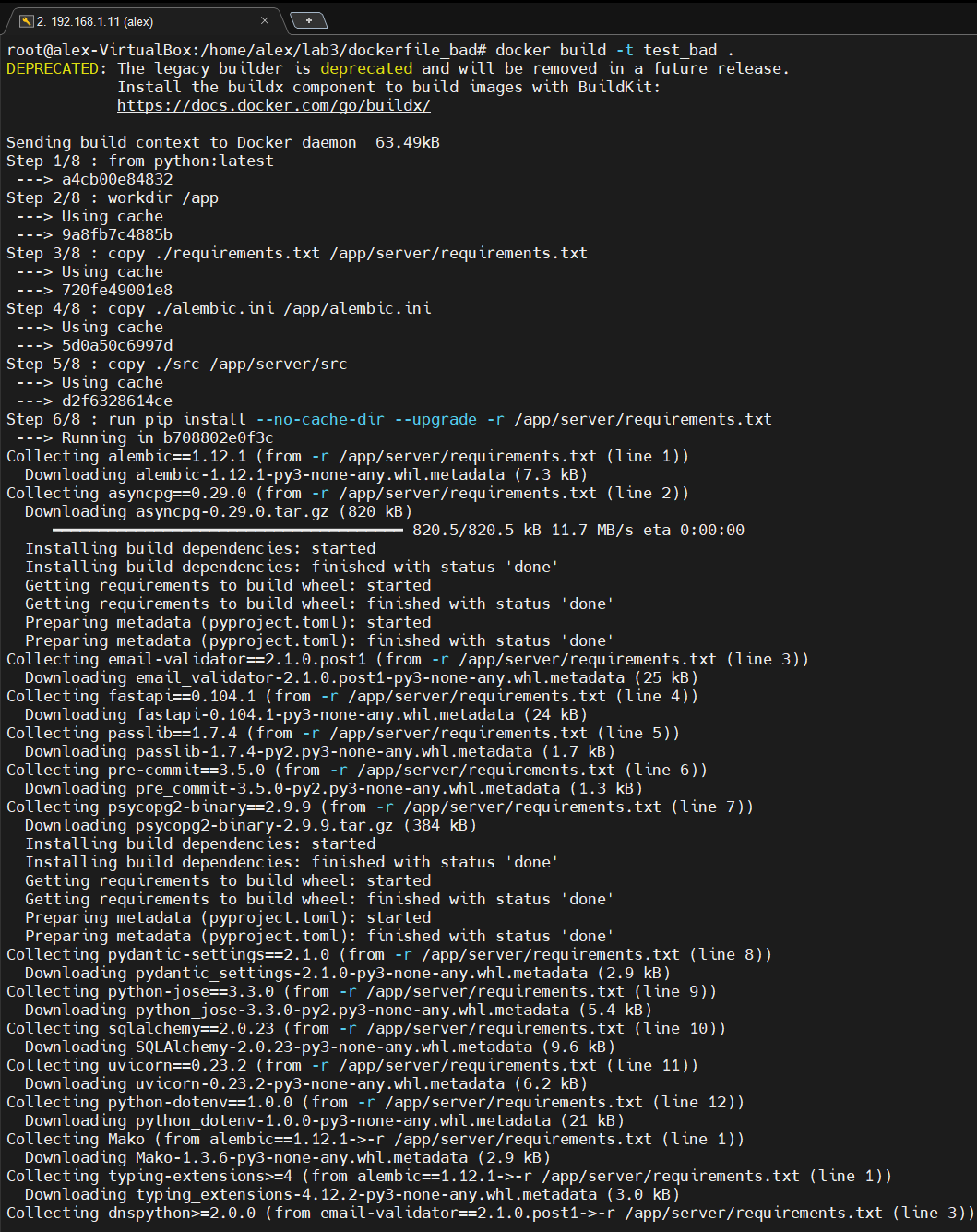


Рисунок 1.4 - Запуск плохого файла

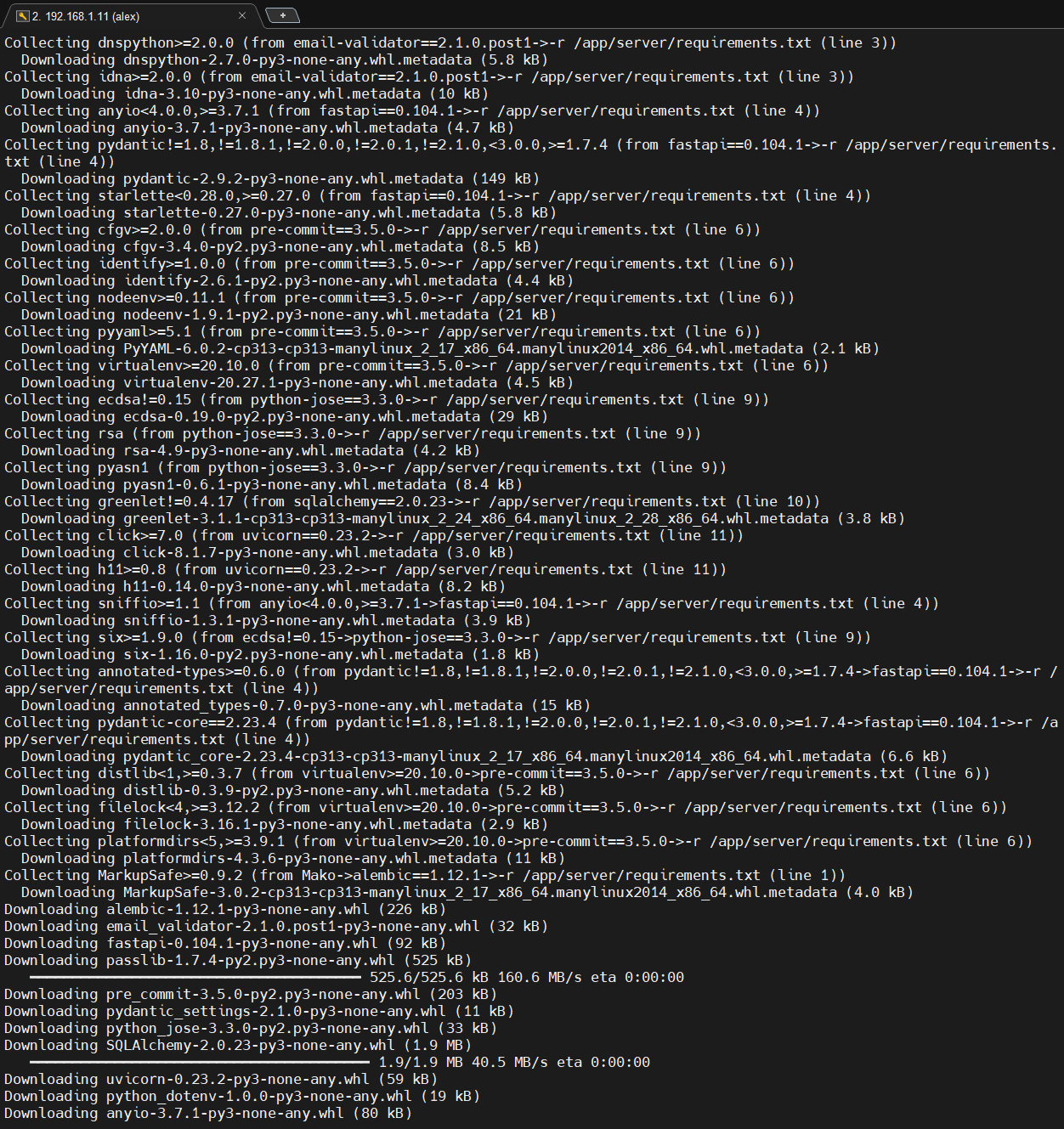


Рисунок 1.5 - Запуск плохого файла

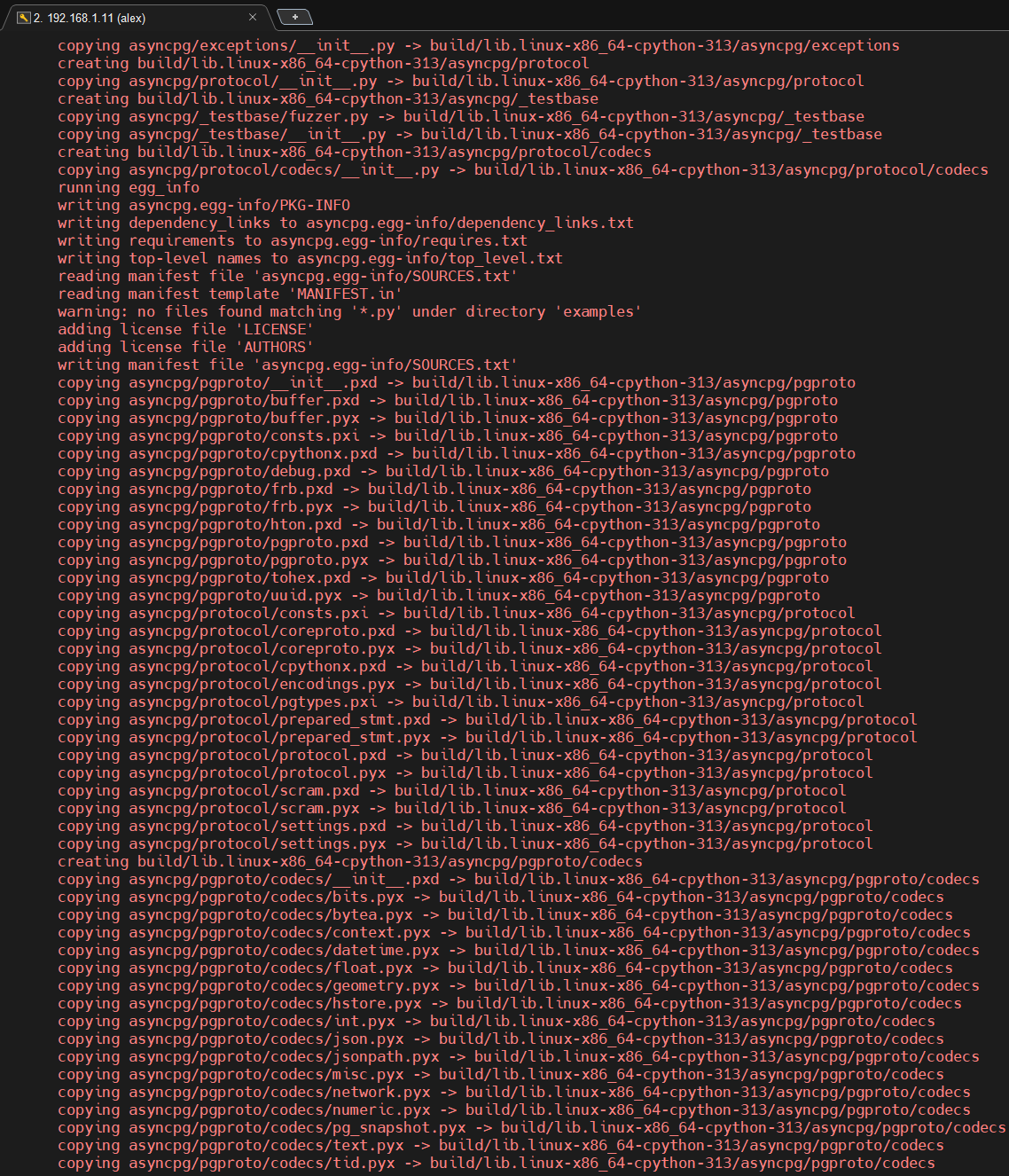


Рисунок 1.6 Запуск плохого файла

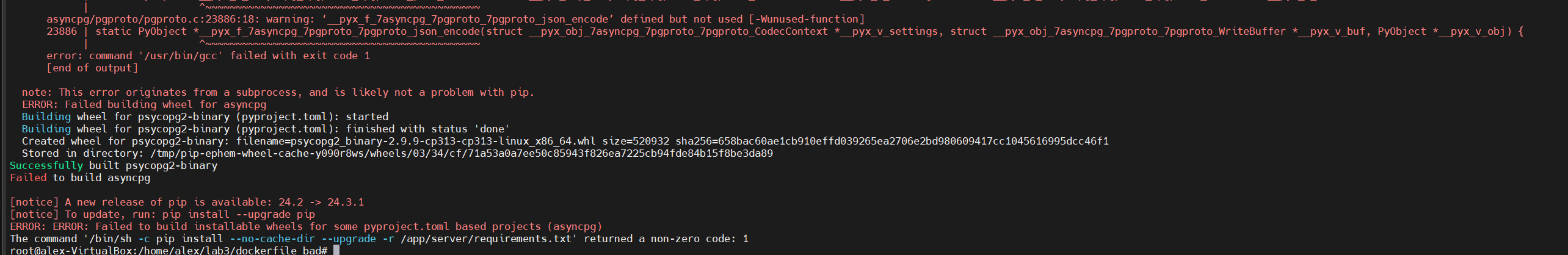


Рисунок 1.7 Запуск плохого файла

После запуска плохого файла получим ошибки, связанные с установкой библиотеки asyncpg.

Запустим хороший dockerfile.

# sudo docker build -t test\_good .

**docker run -d --name test\_ good -p 8000:8000 test\_ good**

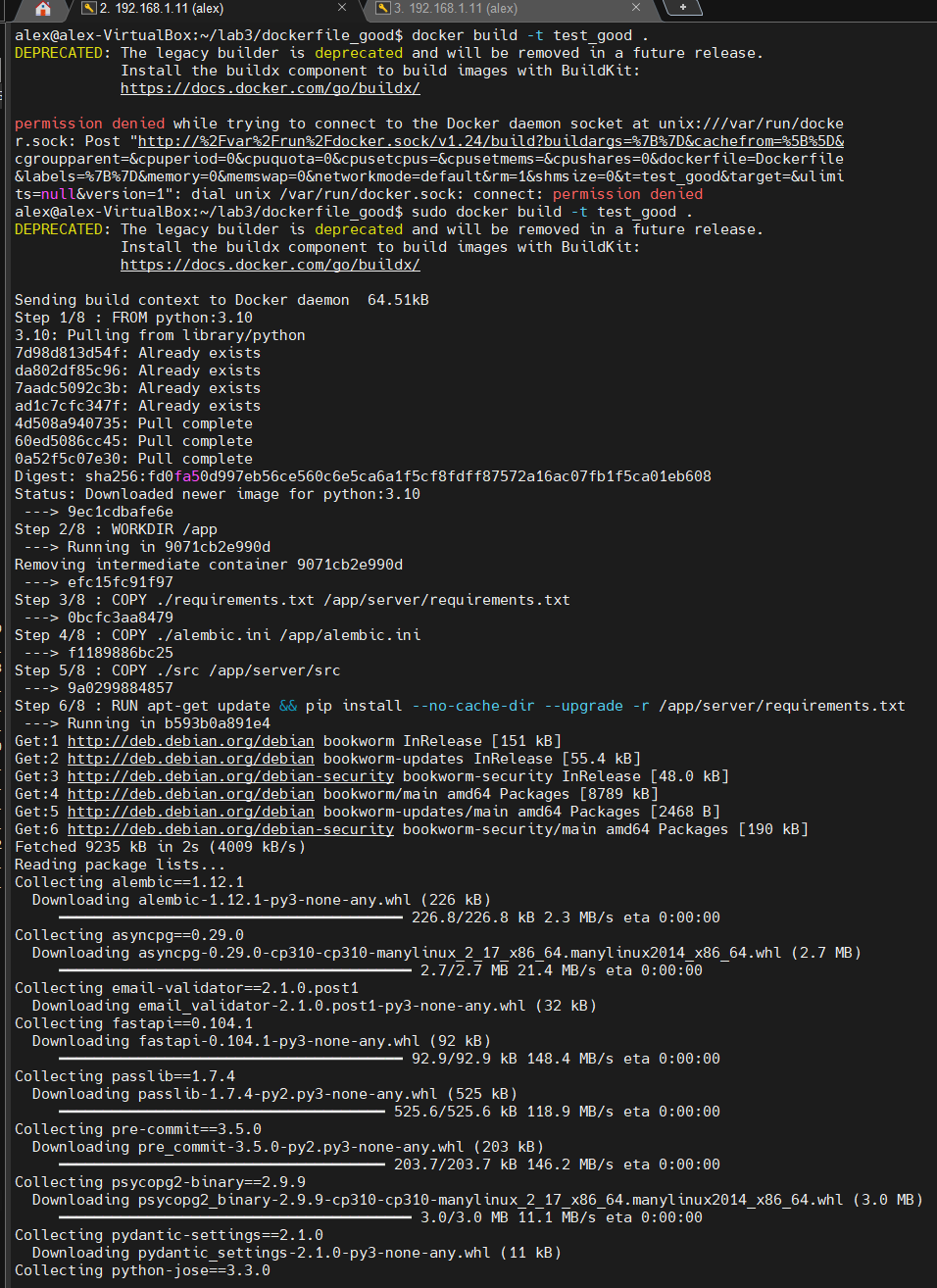


Рисунок 1.7 Запуск хорошего файла

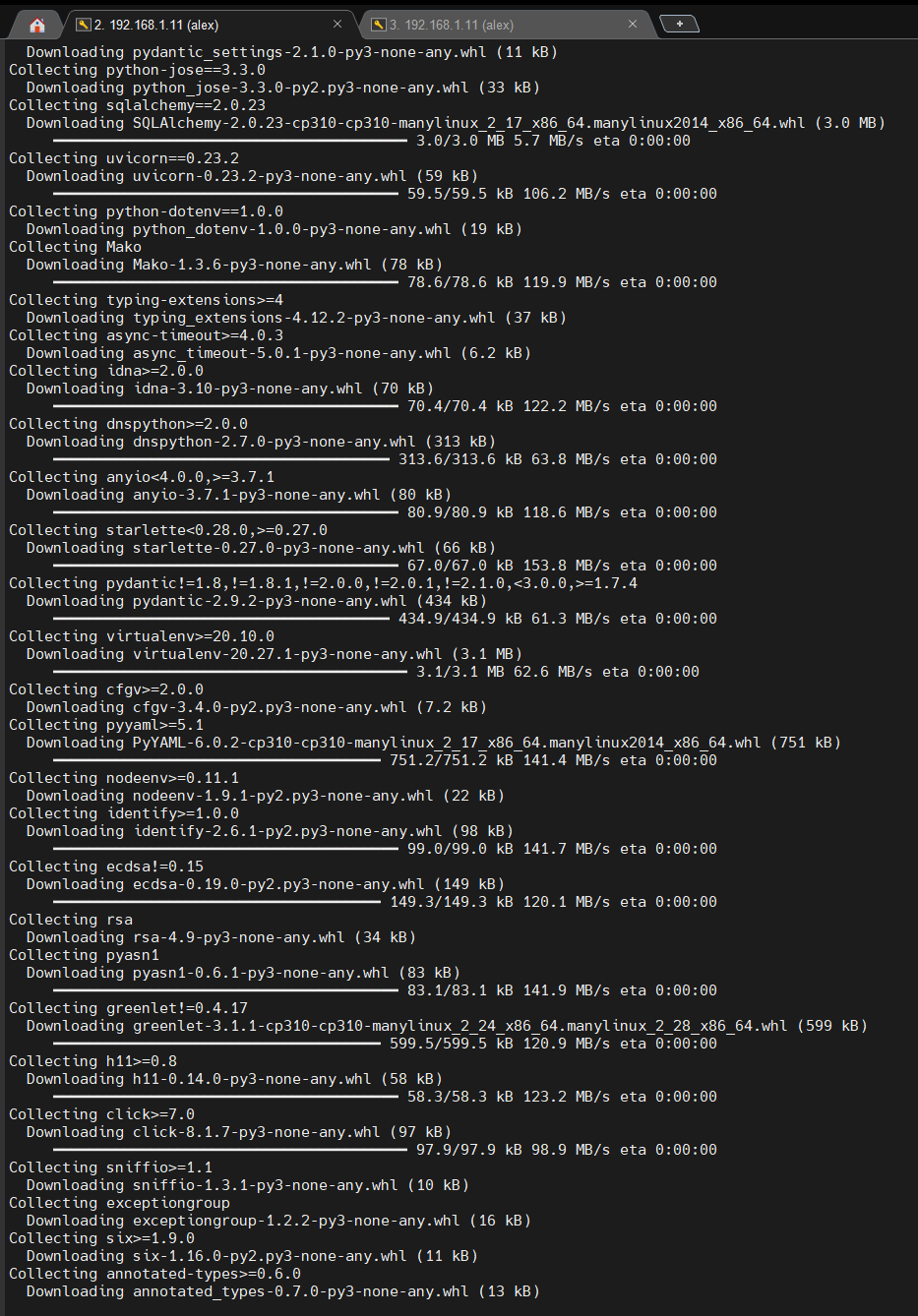
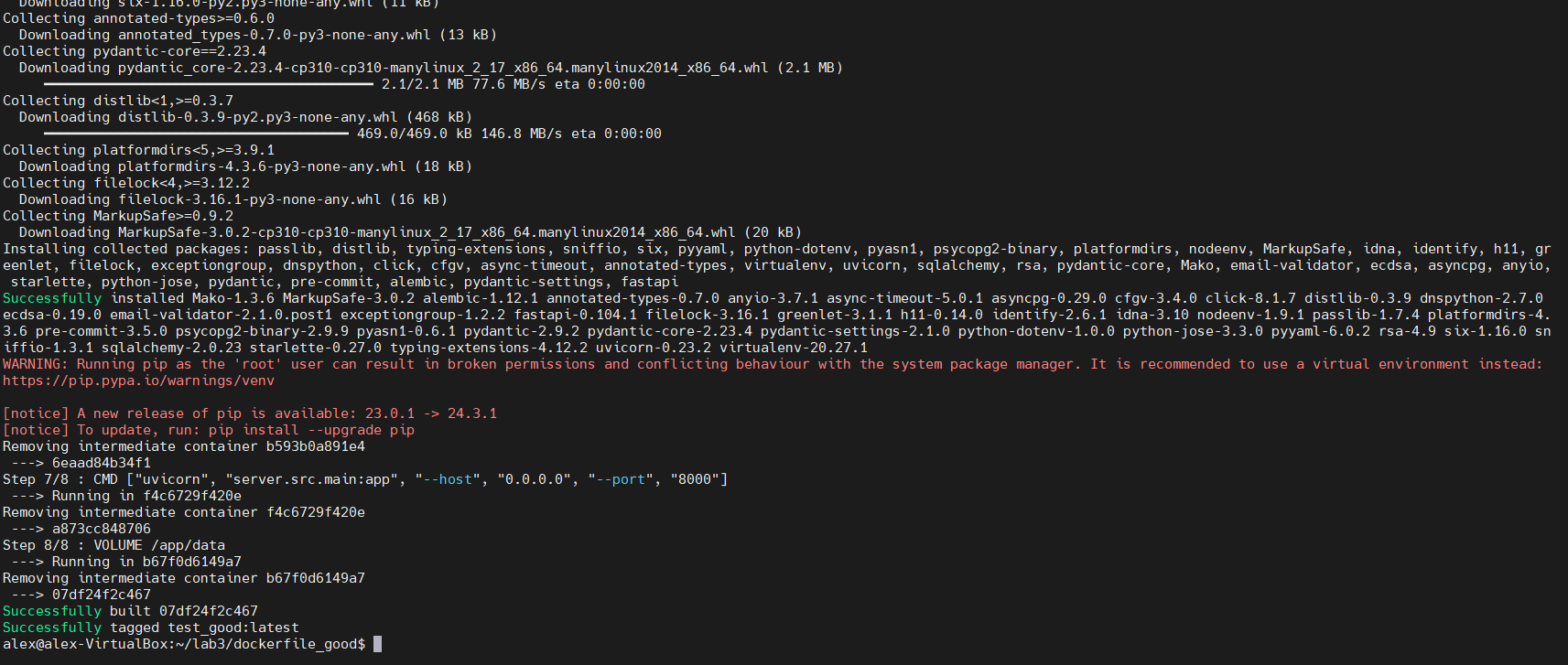


Рисунок 1.8 Запуск хорошего файла



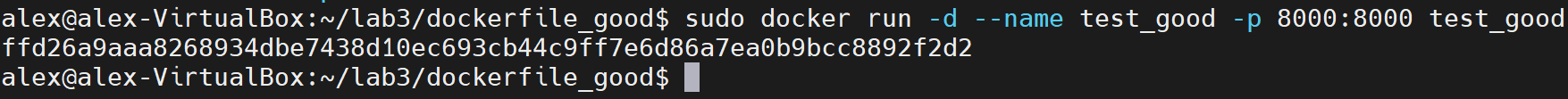


Рисунок 1.9 Запуск хорошего файла

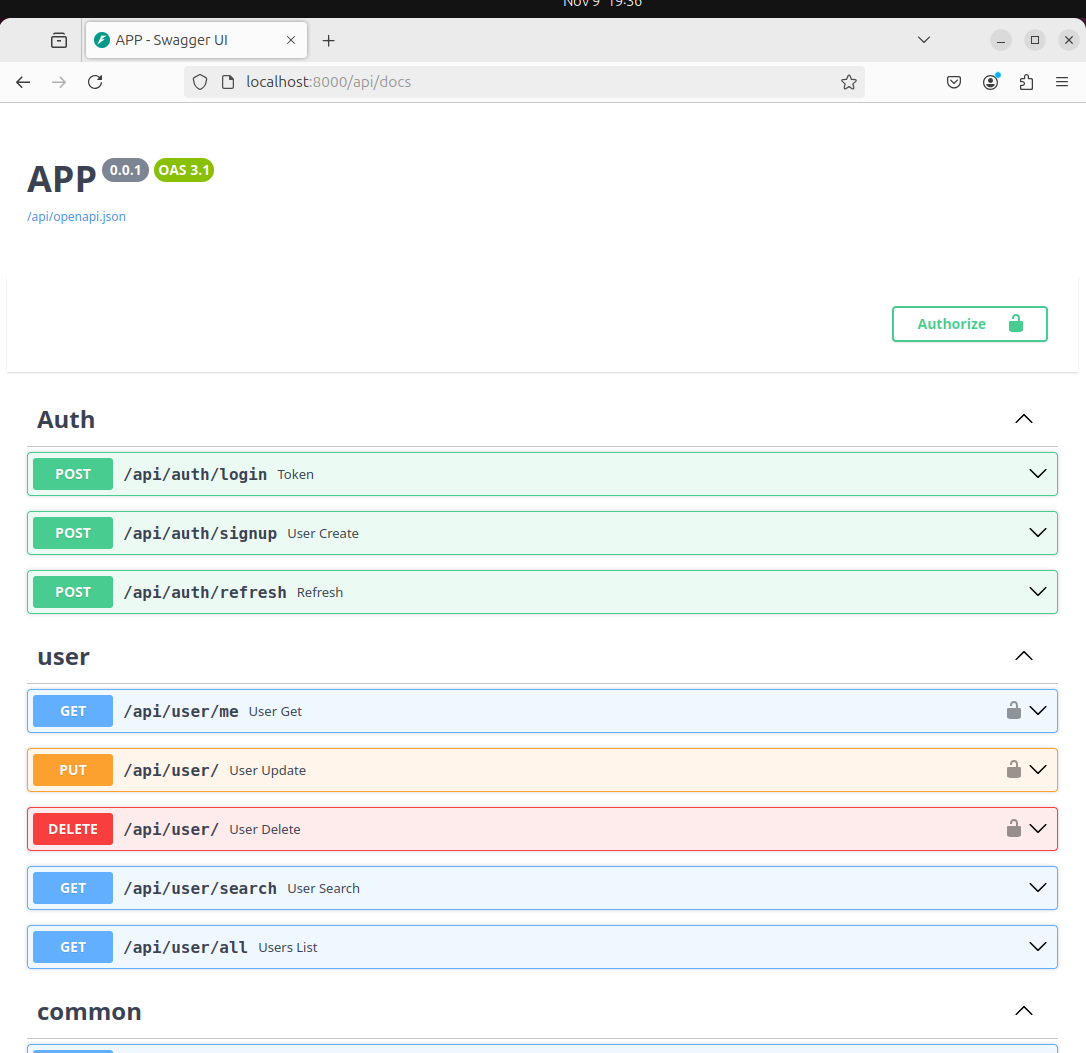


Рисунок 1.10 Результат выполнения хорошего dockerfile

Для просмотра размера хорошего образа пропишем

# sudo docker images

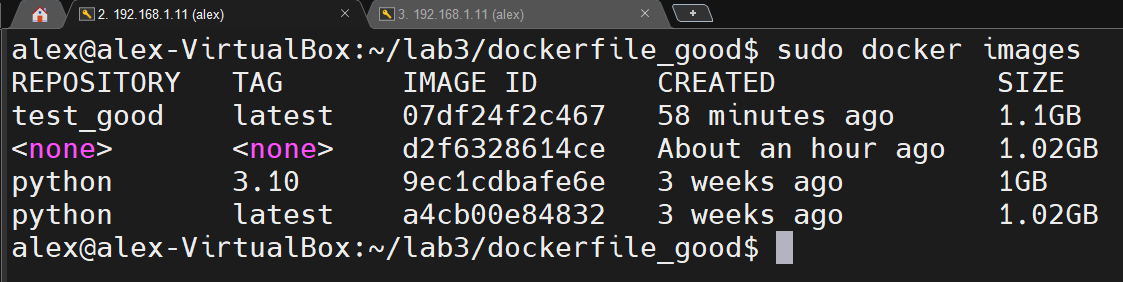


Рисунок 1.11 Результат использования команды

Для просмотра кол-ва слоев хорошего образа пропишем sudo docker **history 07df24f2c467**

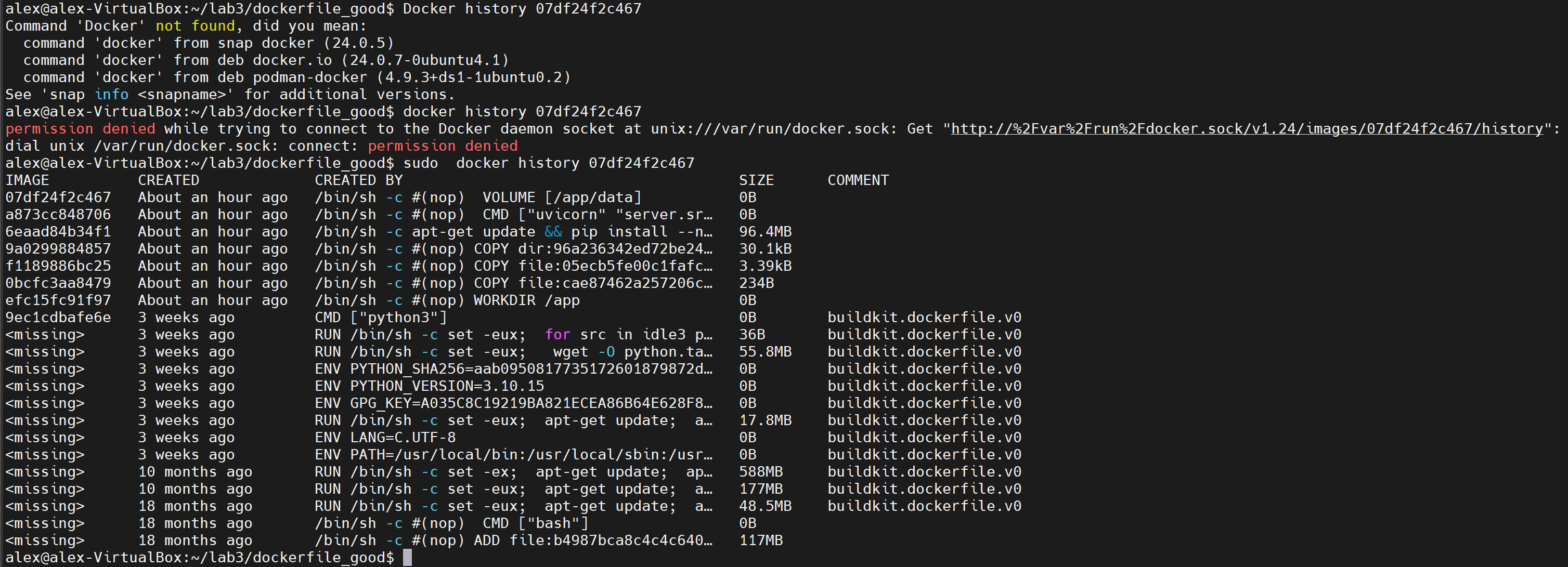


Рисунок 1.12 Количество слоев

Время сборки

Для просмотра времени сборки используем команду

**sudo time docker build -t test\_good . –** для просмотра сборки при использовании кэшированных слоев

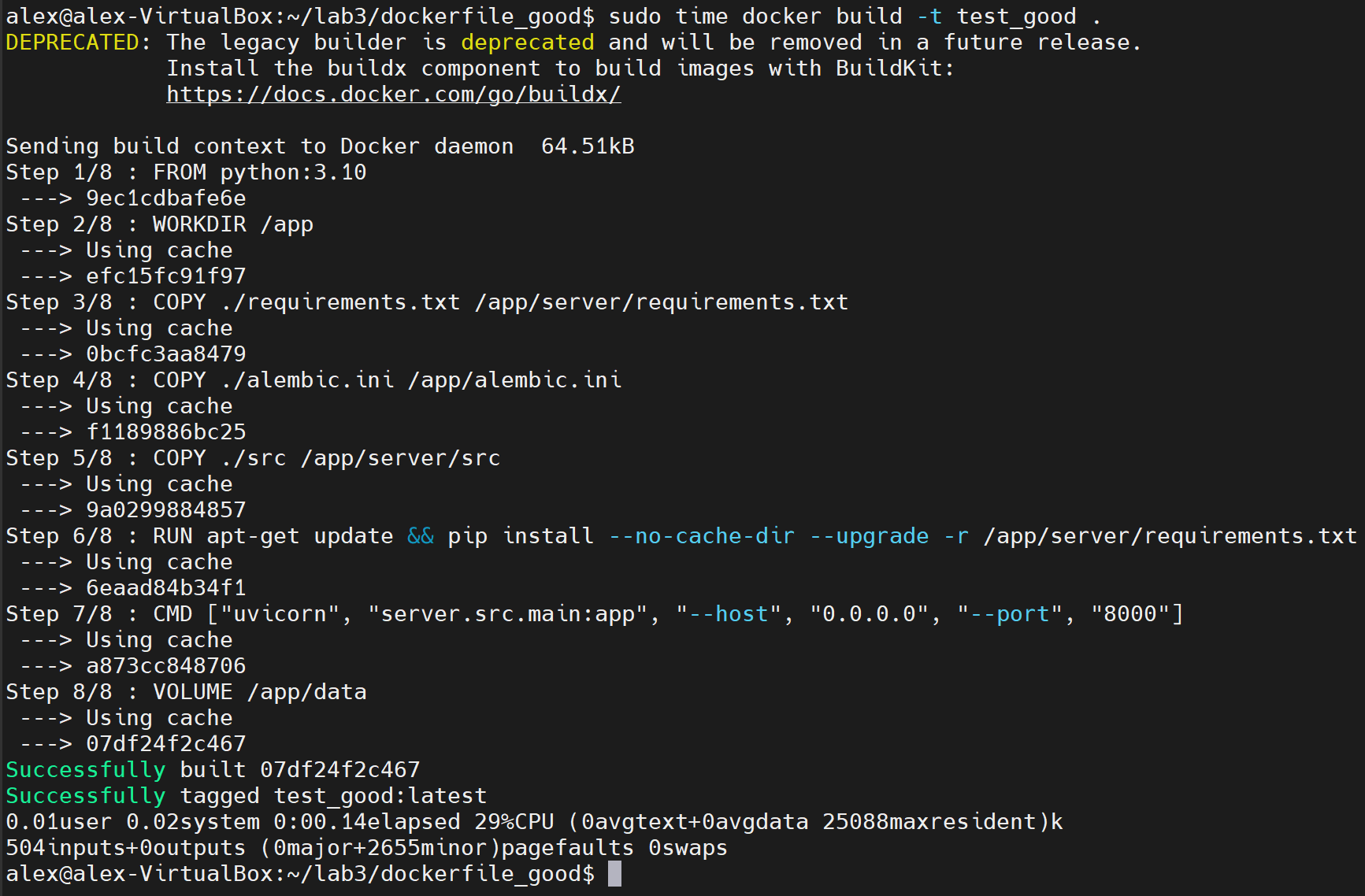


Рисунок 1.13 - Время сборки

При изменение седьмого шага команды 7 и 8 выполнились заново, а остальные были взяты из кэша.

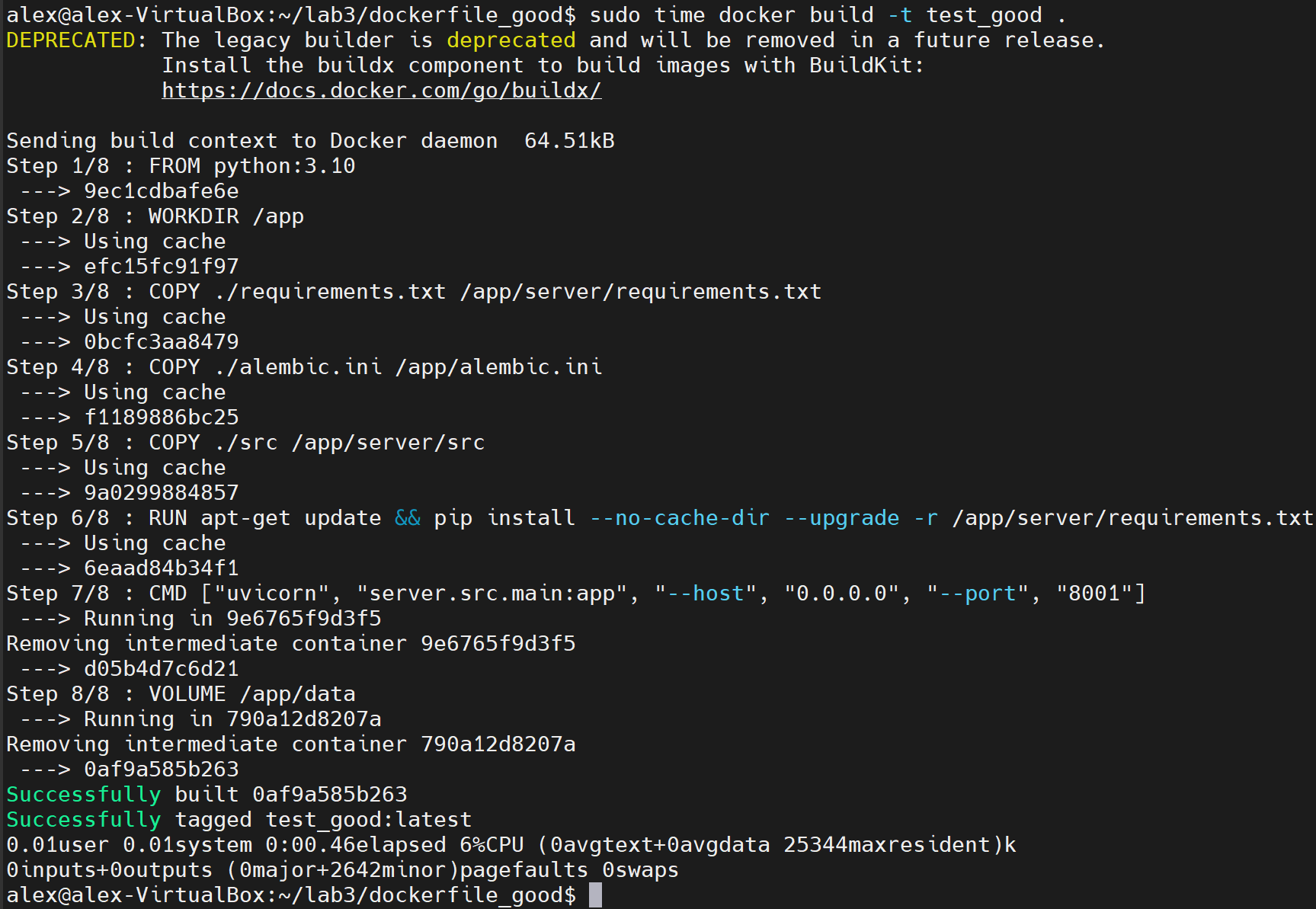


Рисунок 1.14 - Время сборки с измененным портом

Содержание плохого dockerfile

from python:latest workdir /app

copy ./requirements.txt /app/server/requirements.txt copy ./alembic.ini /app/alembic.ini

copy ./src /app/server/src

run pip install --no-cache-dir --upgrade -r /app/server/requirements.txt

cmd ["uvicorn", "server.src.main:app", "--host", "0.0.0.0", "--port", "8000"] volume /app/data

Содержание хорошего dockerfile

FROM python:3.10 WORKDIR /app

COPY ./requirements.txt /app/server/requirements.txt COPY ./alembic.ini /app/alembic.ini

COPY ./src /app/server/src

RUN apt-get update && pip install --no-cache-dir --upgrade -r

/app/server/requirements.txt

CMD ["uvicorn", "server.src.main:app", "--host", "0.0.0.0", "--port", "8000"] VOLUME /app/data

# Анализ различий между подходами

Анализ различий между подходами:

В неудачном Dockerfile используется тег latest для Python, что не позволяет контролировать версию образа. Это может привести к изменениям и несовместимостям, если образ с тегом latest будет обновлен. Также RUN pip install устанавливает зависимости без обновления, что может вызвать проблемы с зависимостями. Перед установкой зависимостей следует обновить пакетный менеджер командой RUN apt-get update && pip install.

# Заключение

После выполнения практической работы, были получены практические и теоретические навыки по работе с Docker и Dockerfile, рассмотрены различные практики по использованию контейнеров.