

**СОДЕРЖАНИЕ**

[**1. ПОСТАНОВКА ЗАДАЧИ** 2](#_Toc196839713)

[**2. ОСНОВНАЯ ЧАСТЬ** 3](#_Toc196839714)

[**3. ВЫВОДЫ** 14](#_Toc196839715)

# **1. ПОСТАНОВКА ЗАДАЧИ**

В практической работе необходимо выполнить все шаги из разделов 1–7. В отчёт должны быть включены ответы на вопросы, выделенные курсивом, результаты выполнения команд из разделов 1–7, а также выполненное индивидуальное задание (раздел 8): листинг Dockerfile, а также команды сборки и запуска контейнера.

Мой номер студента Албахтин И.В. по списку в группе равен N=3. Максимальный номер задания в таблице (число вариантов) M=15. Тогда номер задания V студента Албахтина И.В. будет равен остатку от деления V=Ост((N-1)/M)+1=Ост((3-1)/15)+1= Ост(2/15)+1= 2+1= 3

Индивидуальный вариант № 3:

|  |  |
| --- | --- |
| Вариант | Устанавливаемый пакет |
| 3 | zip |

# **2. ОСНОВНАЯ ЧАСТЬ**

**2.1 Образы:**

Проверим наличие имеющихся образов Docker:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

**Рисунок 2.1 — Проверка наличия образов Docker**

Как мы видим, репозиторий пуст. Загрузим образ Ubuntu с помощью соответствующей команды:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.2 — Загрузка образа Ubuntu**

Проверим наличие образов еще раз:

Изображение выглядит как текст, Шрифт, снимок экрана, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.3 — Проверка наличия образов Docker**

Посмотрим список контейнеров с помощью соответствующей команды:

Изображение выглядит как снимок экрана, текст, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.4 — Список существующих контейнеров(пуст)**

**2.2 Изоляция:**

Посмотрим информацию о хостовой системе с помощью команды hostname:



**Рисунок 2.5 — Информация о хостовой системе**

Сделаем это еще раз

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.6 — Информация о хостовой системе – повторный запрос**

Вопрос: Одинаковый ли результат получился при разных запусках?

Ответ: Да, так как между запросами не проводилось никаких действий, влияющих на результат

Попробуем запросить информацию о хостовой системе от скачанного образа Ubuntu:



**Рисунок 2.7 — Информация о хостовой системе от Ubuntu**

Попробуем еще раз:

Изображение выглядит как текст, Шрифт, снимок экрана

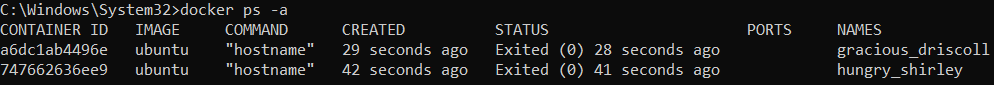
Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.8 — Информация о хостовой системе от Ubuntu – повторный запрос**

Вопрос: Одинаковый ли результат получился при разных запусках?

Ответ: Нет, так как из одного образа ubuntu были запущены два изолированных контейнера

Проверим нашу теорию, еще раз запросив список контейнеров:



**Рисунок 2.9 — Список существующих контейнеров**

Действительно, мы видим два изолированных контейнера

Попробуем запустить bash в контейнере с помощью соответствующей команды:

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.10 — Попытка запуска bash**

В этом случае ничего не произошло, так как интерактивные оболочки выйдут после выполнения любых скриптовых команд, если только они не будут 1 запущены в интерактивном терминале. Попробуем еще раз, отредактировав команду:



**Рисунок 2.11 — Запуск bash**

**2.3 Работа с портами:**

Для начала загрузим образ python с помощью соответствующей команды:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.12 — Загрузка образа python**

В качестве примера запустим встроенный в Python модуль веб-сервера из корня контейнера, чтобы отобразить содержание контейнера:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Рисунок 2.13 — Запуск веб-сервера Python**

Однако, если открыть этот адрес, то ничего не будет видно, потому что порты не проброшены. Запустим сервер еще раз, учтя этот момент:



**Рисунок 2.14 — Запуск веб-сервера Python с перебросом портов**

Проверим корректность работы веб-сервера:

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.15 — Содержимое веб-страницы**

Во время работы сервера изменим его порт с помощью специальной команды:



**Рисунок 2.16 — Изменение порта сервера**

Проверим внесенные изменения:

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.17 — Новый порт сервера**

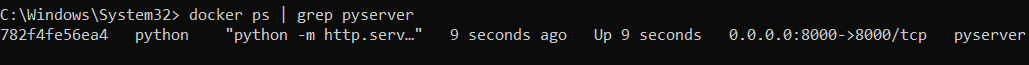
**2. 4 Именованные контейнеры, остановка и удаление:**

Запустим контейнер веб-сервера и переведем в фоновый режим с помощью соответствующих команд:



**Рисунок 2.18 — Запуск сервера в фоновом режиме**

Проверим, что сервер все еще работает:



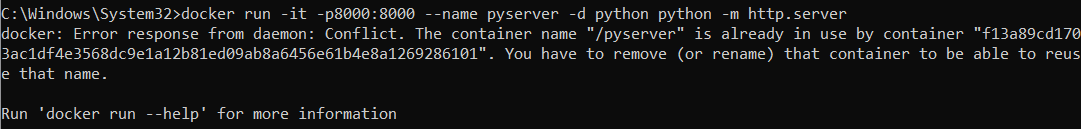
**Рисунок 2.19 — Проверка функционирования сервера**

Остановим процесс работы сервера с помощью соответствующей команды:



**Рисунок 2.20 — Завершение работы сервера**

Однако завершение процесса не означает удаление контейнера. Попытаемся создать контейнер с тем же именем и увидим ошибку:



**Рисунок 2.21 — Попытка создания дубликата сервера**

Чтобы полностью удалить контейнер, воспользуемся командой docker rm:

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.22 — Удаление контейнера pyserver**

**2.5 Постоянное хранение данных:**

Для начала запустим сервер, который будет отображать информацию о файлах в директории /mnt



**Рисунок 2.23 — Запуск pyserver**

Вопрос: Что значат остальные флаги запуска? Где здесь команда, которая выполнится в контейнере?

Ответ: Флаги запуска:

--rm – процесс удалится поле завершения.

--name – выдача имени контейнеру

Попадем в контейнер с помощью соответствующей команды и создадим текстовый файл:

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.24 — Создание текстового файла**

Проверим, появился ли созданный файл в контейнере:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.25 — Страница веб-сервера**

Остановим контейнер и снова создадим. Проверим веб-сервер еще раз:

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.26 — Страница веб-сервера после перезапуска**

Как мы видим, файл исчез – после остановки сервера он удалился, так как не был сохранен. Для сохранения файлов существуют два способа.

**2.5.1 Способ №1:**

Первый способ — это создать отдельный том с помощью ключа -v myvolume:/mnt, где myvolume — название тома, /mnt — директория в контейнере, где будут доступны данные.



**Рисунок 2.27 — Создание контейнера с примонтированным томом**

Затем создадим файл тем же способом:

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.28 — Создание текстового файла**

Убедимся в том, что файл сохранился



**Рисунок 2.29 — Путь до тома с файлом**

**2.5.2 Способ №2:**

Второй способ - монтирование директорий и файлов.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

**Рисунок 2.30 — Создание текстового файла**

Запустим контейнер с помощью соответствующей команды:



**Рисунок 2.31 — Запуск конвейера**

С помощью bash просмотрим содержимое директории /mnt.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.32 — Содержимое директории /mnt**

Как мы видим, там содержится файл hi.txt.

**2.6 Переменные окружения:**

Для передачи переменных окружения внутрь контейнера используется ключ -e. Например, чтобы передать в контейнер переменную окружения MIREA во значение «ONE LOVE», нужно добавить ключ -e MIREA="ONE LOVE".

Проверим, выведя все переменные окружения, определённые в контейнере с помощью утилиты:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.33 — Вывод всех переменных окружений**

**2.7 Dockerfile:**

Соберем образ, в который будут установлены дополнительные пакеты, примонтируем директорию и установим команду запуска с помощью последовательности команд:



**Рисунок 2.34 — Создание файла Dockrefile**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание**

**Рисунок 2.35 — Заполнение файла Dockrefile**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.36 — Успешный build**

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.37 — Успешный запуск**

**2.8 Индивидуальный вариант:**

|  |  |
| --- | --- |
| Вариант | Устанавливаемый пакет |
| 3 | zip |

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.38 — Содержимое Dockerfile**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.39 — Успешный build**

**Изображение выглядит как снимок экрана, текст, Шрифт

Автоматически созданное описание**

**Рисунок 2.39 — Успешный запуск**

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.40 — Запущенный сервер**

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2.41 — Содержимое файла student.txt**

# **3. ВЫВОДЫ**

Было произведено знакомство с программой Docker, управлением контейнерами и взаимодействием с прочими атрибутами системы.