|  |
| --- |
|  |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий  Кафедра Математического обеспечения и стандартизации информационных технологий  **Отчет по практической работе №3**  по дисциплине «Технология разработки программных приложений»  по теме «Docker»   |  |  | | --- | --- | | **Выполнил:**  Студент группыИКБО-20-22 | Шумахер М. Е. | | **Проверил:** | преподаватель Мельников Д. А. |   МОСКВА 2024 г. |

**СОДЕРЖАНИЕ**

[Задание 3](#_Toc160900217)

[Выполнение заданий 4](#_Toc160900218)

[Вывод 8](#_Toc160900219)

Задание

**Цель работы**: получить навыки по работе с Docker’ом.

**Задание для выполнения**

Выполнить все шаги из методических указаний: <https://online-edu.mirea.ru/pluginfile.php?file=%2F1177786%2Fmod_assign%2Fintroattachment%2F0%2FМУ%20ТРПП%20Практика%20№3%20-%20Docker%2BPlayWithDocker.pdf&amp;forcedownload=1>

Выполнение заданий

Посмотрим на имеющиеся образы и загрузим образ ubuntu, затем посмотрим список образов еще раз и проверим список контейнеров (рис. 1).

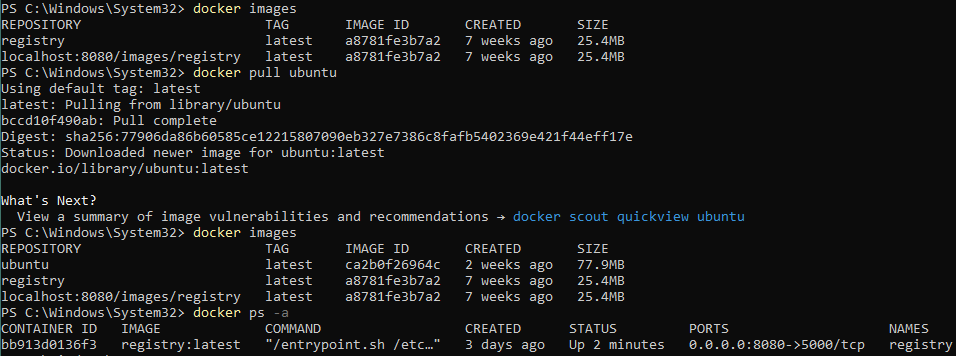


Рисунок 1 – Выполнение команд

Посмотрим информацию о хостовой системе, выполнив команду hostname дважды. Попробуем выполнить то же самое в контейнерах. Выполним два раза команду docker run ubuntu hostname. (рис. 3-4).

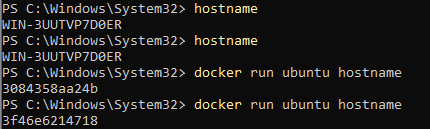


Рисунок 2 – Выполнение команд

Мы получим во втором случае разный результат, это связано с тем, что при каждом запуске контейнера Docker создает новый экземпляр образа Ubuntu, и каждый контейнер может иметь свое собственное уникальное имя хоста. Запустим bash в контейнере: docker run ubuntu bash. Ничего не произошло. Интерактивные оболочки выйдут после выполнения любых скриптовых команд, если только они не будут запущены в интерактивном терминале — поэтому для того, чтобы этот пример не завершился нужно добавить флаги -i -t или сгруппированно -it: docker run -it ubuntu bash. (рис. 3).



Рисунок 3 – Скрипт для считывания содержимого файла

Загрузим образ python командой docker pull python. В качестве примера, запустите встроенный в Python модуль веб-сервера из корня контейнера, чтобы отобразить содержание контейнера. docker run -it python python -m http.server. При запуске пишется, что сервер доступен по адресу http://0.0.0.0:8000/. Однако, если открыть этот адрес, то ничего не будет видно, потому что порты не проброшены. Пробросим порт командой docker run -it -p8000:8000 python python -m http.server. И затем попробуем открыть контейнер на порту 8000 в хостовой системе с портом 8888 docker run -it -p8888:8000 python python -m http.server (рис. 4-5).

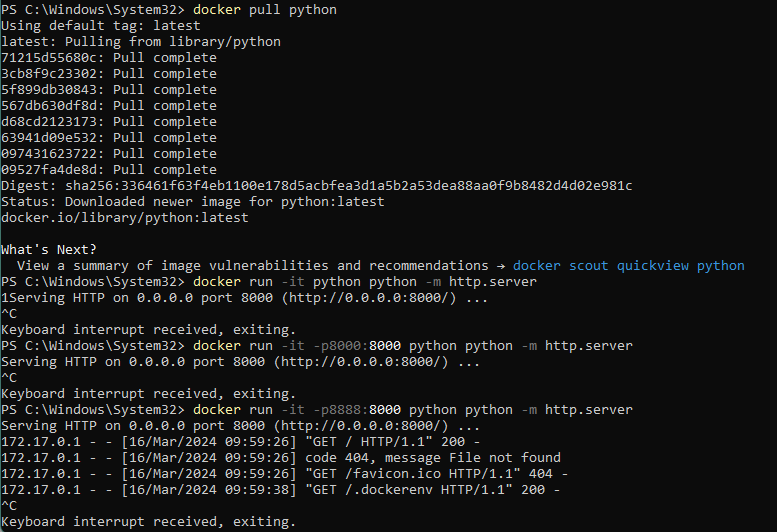


Рисунок 4 – Скрипт для вывода фалов и директорий

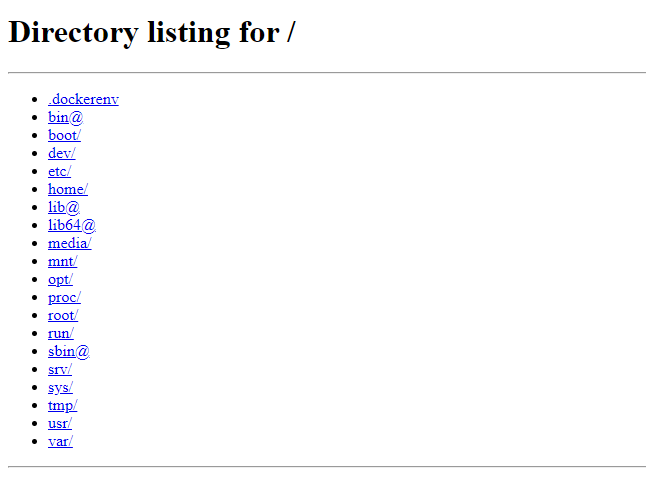


Рисунок 5 – Открывшаяся страница

Для того, чтобы запустить контейнер в фоне, нужно добавить флаг -d/--detach. Также определим имя контейнеру, добавив флаг --name. Убедимся, что контейнер всё ещё запущен: docker ps | grep pyserver — вывод команды не должен быть пустым. Для просмотра логов контейнера, воспользуемся командой docker logs pyserver. Для того, чтобы остановить выполнение контейнера, существует команда docker stop pyserver. Однако, если снова попробовать запустить командой docker run -it -p8000:8000 --name pyserver -d python python -m http.server, то возникнет ошибка: контейнер с таким именем существует. Его нужно удалить docker rm pyserver. Для остановки и удаления контейнера можно воспользоваться командой docker rm -f pyserver вместо выполнения двух отдельных команд stop и rm. После удаления контейнер с таким именем можно будет создать заново. (рис. 6).

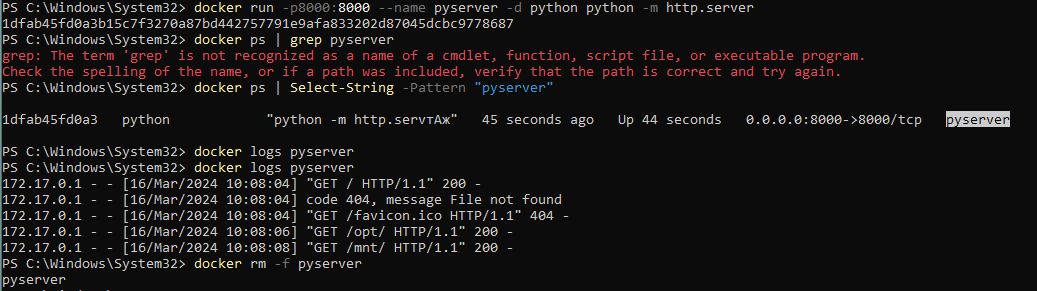


Рисунок 6 – Выполнение команд

Запустите контейнер, в котором веб-сервер будет отдавать содержимое директории /mnt: docker run -p8000:8000 --name pyserver --rm -d python python -m http.server -d /mnt, где -d /mnt указывает модулю http.server какая директория будет корневой для отображения (рис. 7).



Рисунок 7 – Выполнение команды

-p8000:8000: Этот флаг привязывает порт 8000 на вашей хостовой машине к порту 8000 внутри контейнера. --name pyserver: Этот флаг устанавливает имя контейнера как "pyserver". --rm: Этот флаг говорит Docker о том, что контейнер должен быть удален после завершения его работы. Флаг -m используется для указания Python на то, что http.server должен быть запущен как скрипт, а не как модуль. Команда, которая будет выполнена в контейнере, находится в конце команды запуска контейнера: python -m http.server -d /mnt.

Для того, чтобы попасть в уже запущенный контейнер, существует команда docker exec -it pyserver bash — вы попадёте в оболочку bash в контейнере. Попав в контейнер, выполните команду cd mnt && echo "hello world" > hi.txt, а затем выйдите из контейнера, введя команду exit или нажав комбинацию клавиш Ctrl+D (рис. 8). Если открыть http://0.0.0.0:8000/, там будет доступен файл hi.txt.

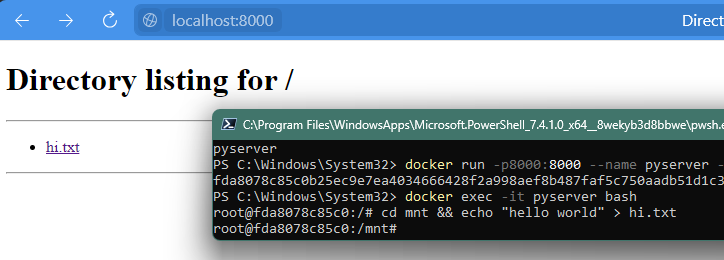


Рисунок 8 – Выполнение команд

Остановим контейнер: docker stop pyserver, а затем снова запустим: docker run -p8000:8000 --name pyserver --rm -d python python -m http.server -d /mnt. (рис. 9).

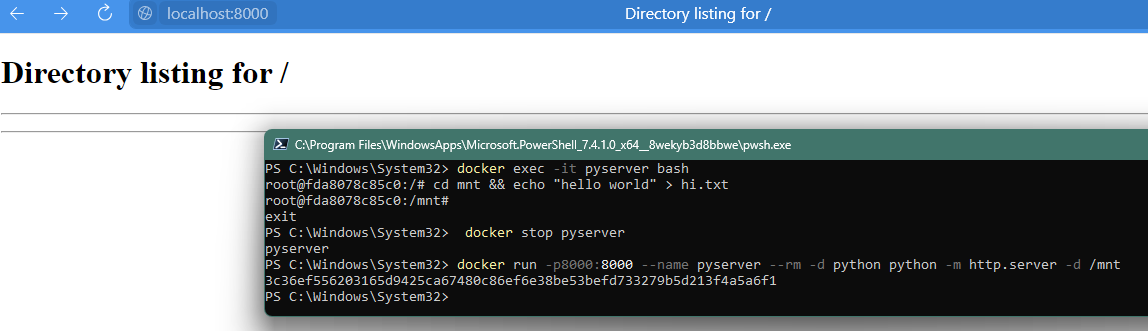


Рисунок 9 – Выполнение команд

Как мы видим, файл hi.txt пропал — это неудивительно, ведь мы запустили другой контейнер, а старый был удалён после завершения работы (флаг --rm). Для того, чтобы не терялись какие-то данные существует механизм монтирования. Первый способ — это создать отдельный том с помощью ключа -v myvolume:/mnt, где myvolume ­– название тома, /mnt – директория в контейнере, где будут доступны данные. Попробуем снова создать контейнер, но уже с примонтированным томом. Затем, если создать файл (выполнить docker exec -it pyserver bash и внутри контейнера выполнить cd mnt && echo "hello world" > hi.txt), то даже после удаления контейнера данные в этом томе будут сохранены. Чтобы узнать где хранятся данные, выполните команду docker inspect -f "{{json .Mounts }}" pyserver, в поле Source будет храниться путь до тома на хостовой машине (рис. 10).

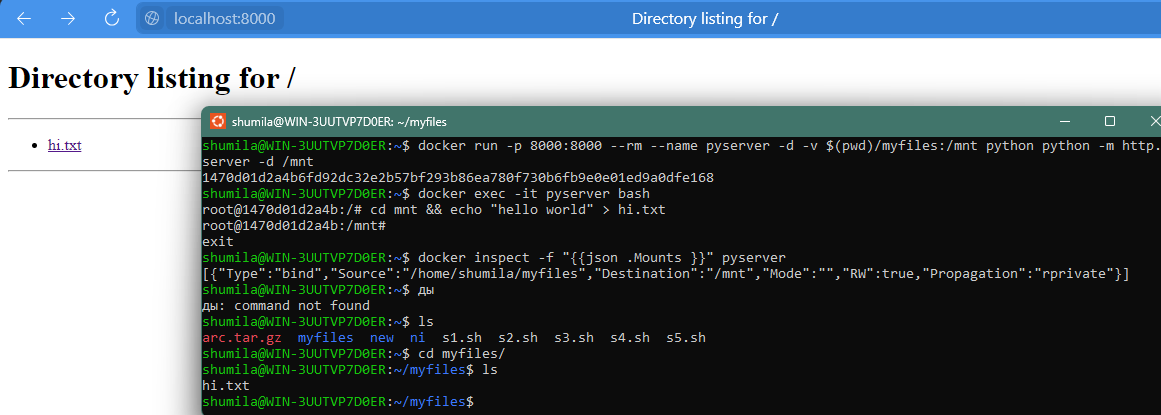


Рисунок 10 – Выполнение команд

Создадим директорию и файлы, которые будем монтировать. Часть из них нам понадобится дальше: создайте директорию: mkdir myfiles, в ней создайте файл host.txt: touch myfiles/host.txt. Запустим контейнер: docker run -p8000:8000 --rm --name pyserver -d -v $(pwd)/myfiles:/mnt python \ python -m http.server -d /mnt. Затем, зайдите в контейнер: docker exec -it pyserver bash, перейдите в директорию /mnt командой cd /mnt. Если вывести список файлов командой ls, то там будет файл host.txt, примонтированный вместе с директорией myfiles (рис. 11).

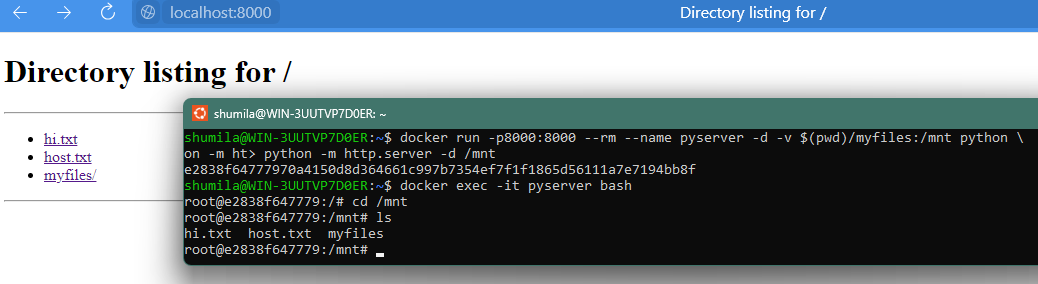


Рисунок 11 – Выполнение команд

Для передачи переменных окружения внутрь контейнера используется ключ -e. Например, чтобы передать в контейнер переменную окружения MIREA со значением «ONE LOVE», нужно добавить ключ -e MIREA="ONE LOVE". Проверьте, выведя все переменные окружения, определённые в контейнере с помощью утилиты env: docker run -it --rm -e MIREA="ONE LOVE" ubuntu env (рис. 12)

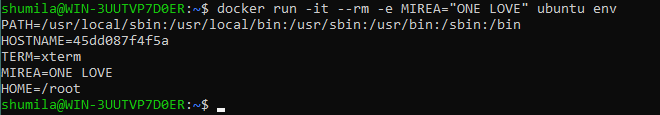


Рисунок 12 – Вывод переменных окружения

Соберите образ, в который будут установлены дополнительные пакеты, примонтируйте директорию и установите команду запуска. Для этого создаётся файл Dockerfile. Соберем образ с тегом mycoolimage с помощью команды docker build -t mycoolimage .

Запустим образ docker run --rm -it -p8099:80 mycoolimage, где порт 8099 – порт на хостовой машине (рис. 13-14).

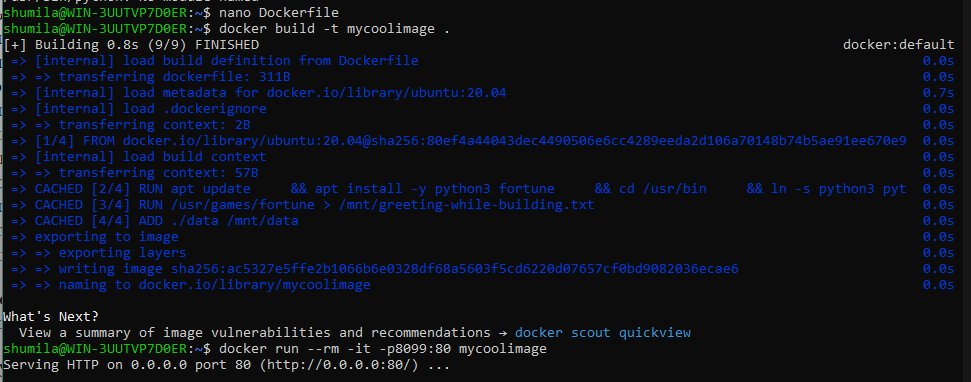


Рисунок 13 – Запуск контейнера

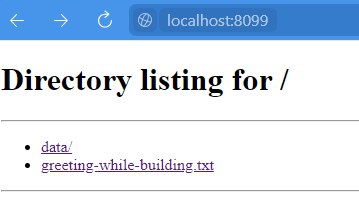


Рисунок 14 – Результат запуска контейнера

Напишем Dockerfile, собрем образ, запустим контейнер. Для монтирования создадим директорию data и в ней файл student.txt, содержащий ФИО, название группы и номер варианта. Согласно варианту: необходимо использовать базовый образ ubuntu:20.04 и примонтировать директорию data в директорию /mnt/files/ в контейнере. Установить пакет, согласно варианту: emacs-nox. Запустить веб-сервер, отображающий содержимое /mnt/files, в хостовой системе должен открываться на порту 8830 (рис. 15-17).

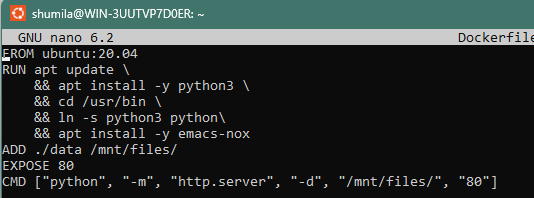


Рисунок 15 – Dockerfile для выполнения задания



Рисунок 16 – Результат запуска контейнера



Рисунок 17 – Содержимое student.txt

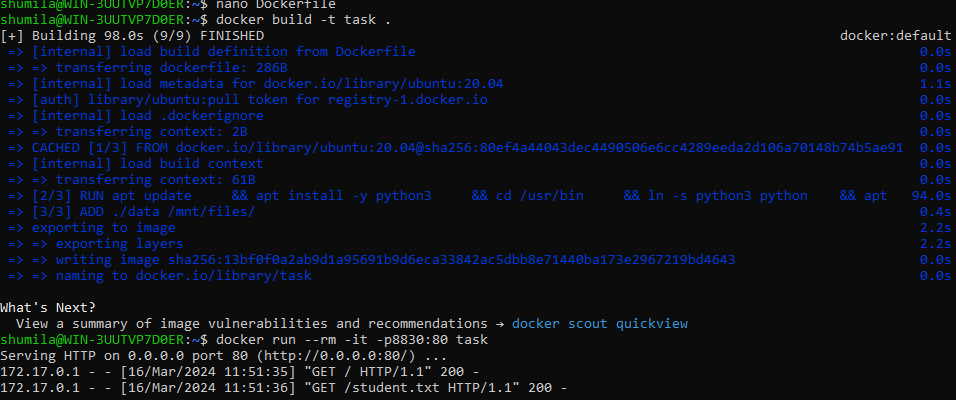


Рисунок 18 – Выполнение команд

Вывод

В результате выполнения работы были получены навыки по работе с Docker.