



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра математического обеспечения и стандартизации
информационных технологий

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ
по дисциплине «Технология разработки программных приложений»

Практическое задание № 4

Студент группы ИНБО-08-22 Самойлов М.М.

(подпись)

Старший
преподаватель *Мельников Д. А.*

(подпись)

Отчет представлен «23» марта 2024г.

Москва 2024 г

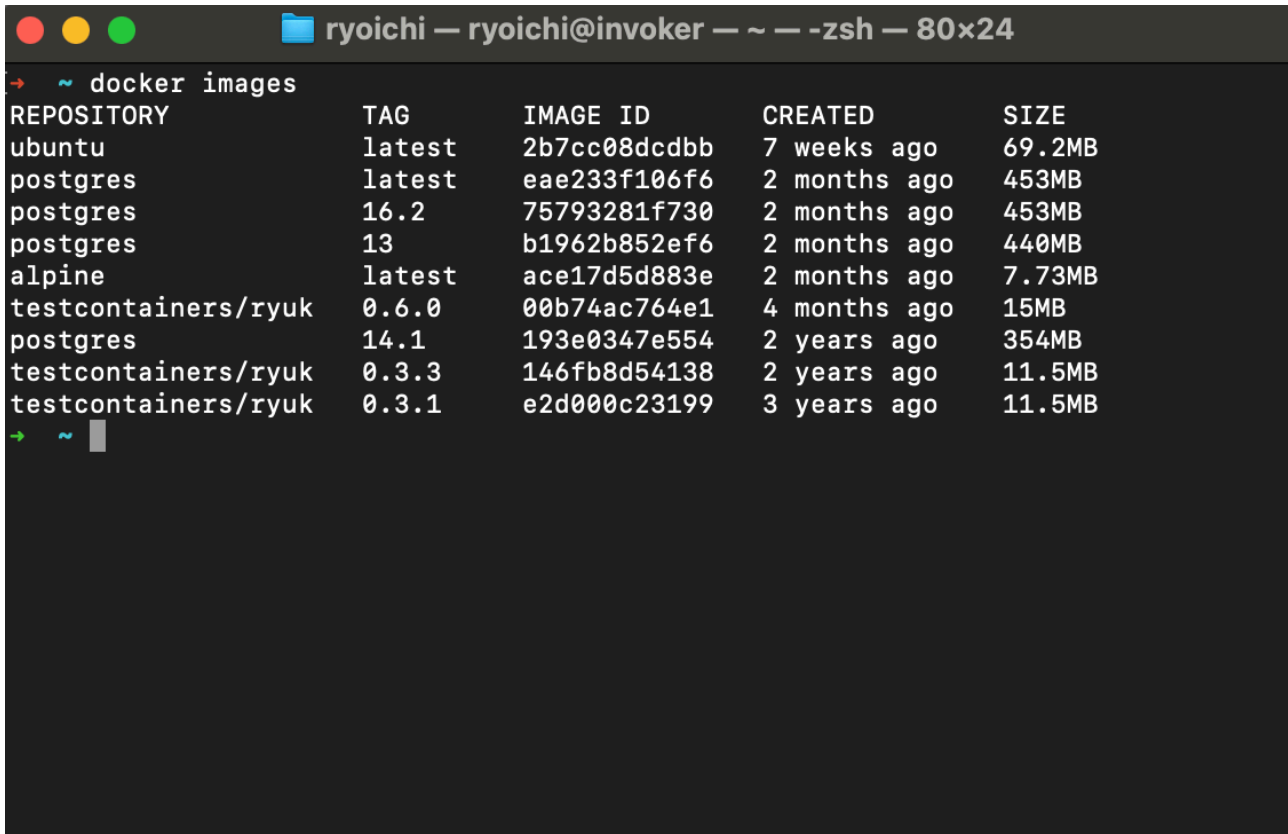
ОГЛАВЛЕНИЕ

Задание.....	Error! Bookmark not defined.
Выполнение заданий	3
Вывод	Error! Bookmark not defined.

Выполнение заданий

1. Образы

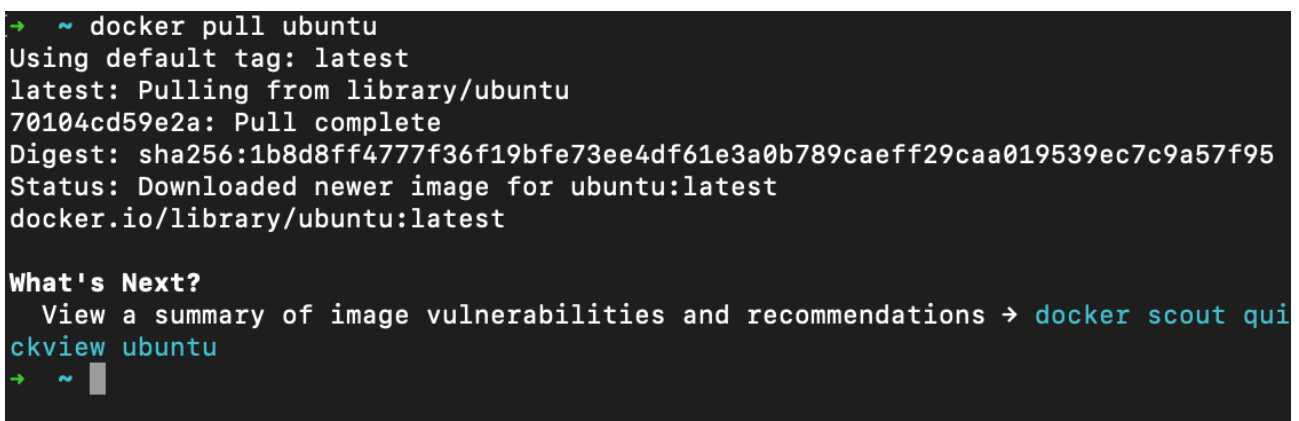
На рисунке 1 отображен запуск команды `docker images` – получение всех имеющихся на устройстве образов



```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
→ ~ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest             2b7cc08dcdbb       7 weeks ago        69.2MB
postgres            latest             eae233f106f6       2 months ago       453MB
postgres            16.2              75793281f730       2 months ago       453MB
postgres            13                b1962b852ef6       2 months ago       440MB
alpine              latest             ace17d5d883e       2 months ago       7.73MB
testcontainers/ryuk 0.6.0              00b74ac764e1       4 months ago       15MB
postgres            14.1              193e0347e554       2 years ago        354MB
testcontainers/ryuk 0.3.3              146fb8d54138       2 years ago        11.5MB
testcontainers/ryuk 0.3.1              e2d000c23199       3 years ago        11.5MB
→ ~ █
```

Рисунок 1 – Запуск команды `docker images`

На рисунке 2 отображен запуск команды `docker pull ubuntu` – скачивание образа ubuntu с докер хаба



```
→ ~ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
70104cd59e2a: Pull complete
Digest: sha256:1b8d8ff4777f36f19bfe73ee4df61e3a0b789caeff29caa019539ec7c9a57f95
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview ubuntu
→ ~ █
```

Рисунок 2 – Запуск команды `docker pull ubuntu`

На рисунке 3 отображен запуск команды `docker images` после скачивания образа

```

→ ~ docker images
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE
ubuntu              latest      dfbcc2701b93 12 days ago  69.2MB
ubuntu              <none>     2b7cc08dcdbb 7 weeks ago  69.2MB
postgres            13          b1962b852ef6 2 months ago 440MB
postgres            16.2        75793281f730 2 months ago 453MB
postgres            latest      eae233f106f6 2 months ago 453MB
alpine              latest      ace17d5d883e 2 months ago 7.73MB
testcontainers/ryuk 0.6.0       00b74ac764e1 4 months ago 15MB
postgres            14.1        193e0347e554 2 years ago  354MB
testcontainers/ryuk 0.3.3       146fb8d54138 2 years ago  11.5MB
testcontainers/ryuk 0.3.1       e2d000c23199 3 years ago  11.5MB
→ ~ █

```

Рисунок 3 – Запуск команды docker images после скачивания образа ubuntu

На рисунке 4 отображен запуск команды docker ps – отображение всех контейнеров

```

→ ~ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS
cee9d559d191   postgres      "docker-entrypoint.s..." 3 weeks ago    Exited (0)
3 weeks ago                                spring-postgres
c9b4bba3552b   75793281f730  "docker-entrypoint.s..." 6 weeks ago    Exited (0)
5 days ago                                some-postgres
→ ~ █

```

Рисунок 4 – Запуск команды docker ps

2. Изоляция

На рисунке 5 отображен запуск команды hostname несколько раз локально. Результат при нескольких вызовах одинаковый.

```

ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
→ ~ hostname
invoker
→ ~ hostname
invoker
→ ~ █

```

Рисунок 5 – Несколько запусков команды hostname локально

На рисунке 6 отображен запуск команды hostname несколько раз в докер-контейнере Ubuntu. Результат при нескольких вызовах разный.

```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
→ ~ docker run -it ubuntu
root@0473168b462f:/# hostname
0473168b462f
root@0473168b462f:/# exit
exit
→ ~ docker run -it ubuntu
root@bec6d98b92d3:/# hostname
bec6d98b92d3
root@bec6d98b92d3:/# exit
exit
→ ~
```

Рисунок 6 – Несколько запусков команды hostname в докер-контейнере Ubuntu

На рисунке 7 отображен запуск команды docker ps

```
→ ~ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS   NAMES
bec6d98b92d3   ubuntu    "/bin/bash"             57 seconds ago   Exit
ed (0) 50 seconds ago          recursing_bell
0473168b462f   ubuntu    "/bin/bash"             About a minute ago   Exit
ed (0) About a minute ago      exciting_borg
6c014a3331ac   ubuntu    "hostname"              4 minutes ago     Exit
ed (0) 4 minutes ago           angry_lovelace
34b6d7b38f91   ubuntu    "hostname"              4 minutes ago     Exit
ed (0) 4 minutes ago           nice_ardinghelli
cee9d559d191   postgres  "docker-entrypoint.s..." 3 weeks ago       Exit
ed (0) 3 weeks ago             spring-postgres
c9b4bba3552b   75793281f730 "docker-entrypoint.s..." 6 weeks ago        Exit
ed (0) 5 days ago              some-postgres
→ ~
```

Рисунок 7 – Запуск команды docker ps

На рисунке 8 отображен запуск контейнера без входа в интерактивный режим. В этом случае не происходит переход в bash-терминал.

```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
→ ~ docker run ubuntu bash
→ ~ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED        STATUS        PORTS   NAMES
→ ~
```

Рисунок 8 – Запуск команды docker run ubuntu bash.

На рисунке 9 отображен запуск контейнера со входом в интерактивный режим. В этом случае происходит переход в bash-терминал.

```
→ ~ docker run -it ubuntu bash
root@4c5ac37b41ea:/# hostname
4c5ac37b41ea
root@4c5ac37b41ea:/# exit
exit
→ ~
```

Рисунок 9 – Запуск команды docker run ubuntu bash.

3. Работа с портами

На рисунке 10 отображен процесс скачивания образа python

```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24

→ ~ docker pull python
Using default tag: latest
latest: Pulling from library/python
1e92f3a395ff: Pull complete
374850c6db17: Pull complete
421c44fab18b: Pull complete
b9717a38adec: Pull complete
51795e508cf7: Pull complete
bc54e015d093: Pull complete
3822d8fd7491: Pull complete
4fc85492ad0c: Pull complete
Digest: sha256:e0e2713ebf0f7b114b8bf9fbcaba9a69ef80e996b9bb3fa5837e42c779dcdc0f
Status: Downloaded newer image for python:latest
docker.io/library/python:latest

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview python
→ ~
```

Рисунок 10 – Скачивание образа python

На рисунке 11 отображен процесс запуска python веб-сервера на порту 8080 без пробрасывания портов наружу. Без этого сервис не будет доступен из вне.

```
ryoichi — docker run -it python python -m http.server — docker — com.do...

→ ~ docker run -it python python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Рисунок 11 – Запуск python веб-сервера на порту 8080 без пробрасывания портов наружу

На рисунке 12 отображен результат запуска python веб-сервера на порту 8080 без пробрасывания портов наружу

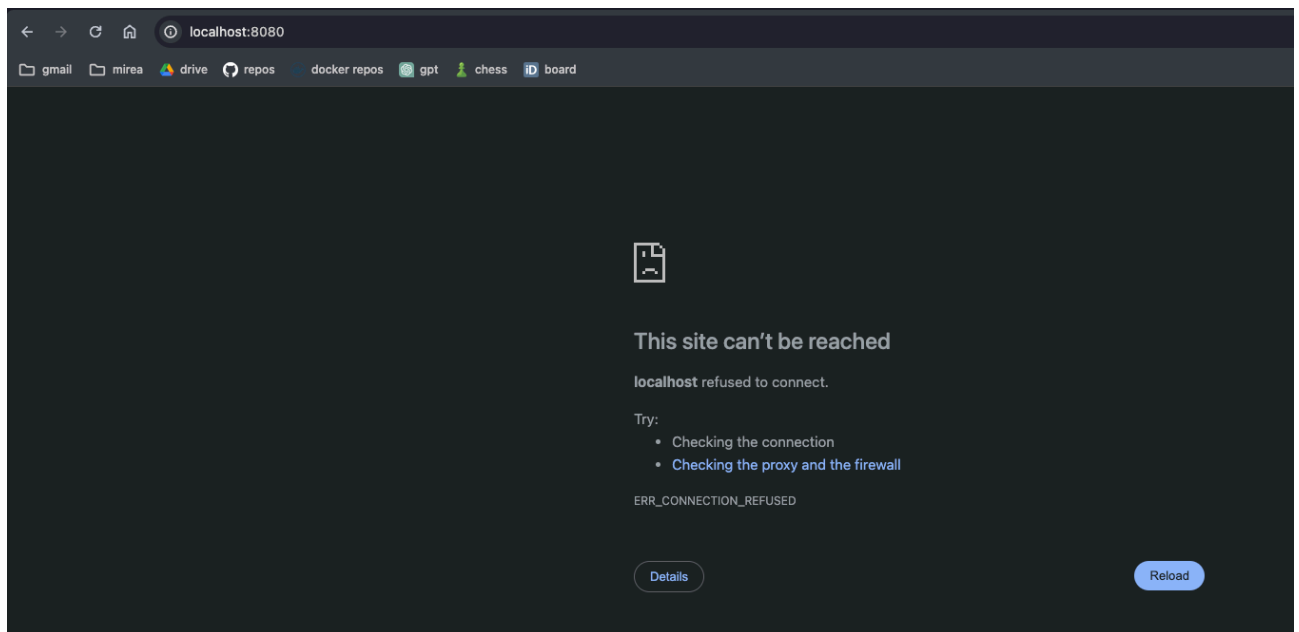


Рисунок 11 – Результат запуска python веб-сервера на порту 8080 без пробрасывания портов наружу

На рисунке 13 отображен процесс запуска python веб-сервера на порту 8080 с пробрасыванием портов наружу.

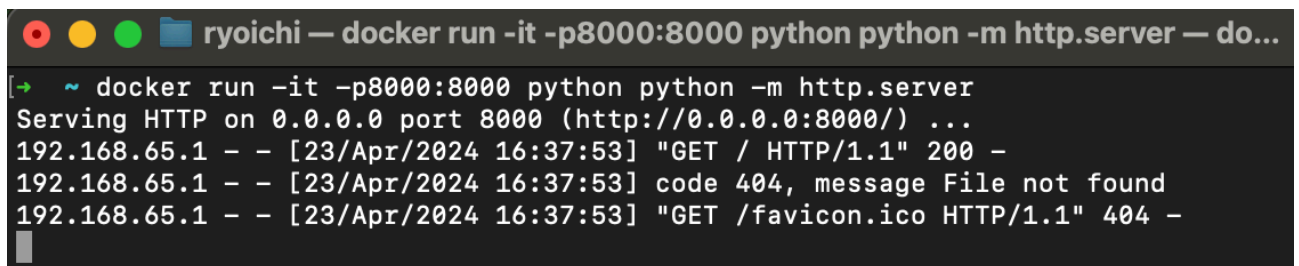


Рисунок 13 – Запуск python веб-сервера на порту 8000 с пробрасыванием портов наружу

На рисунке 14 отображен результат запуска python веб-сервера на порту 8080 с пробрасыванием портов наружу

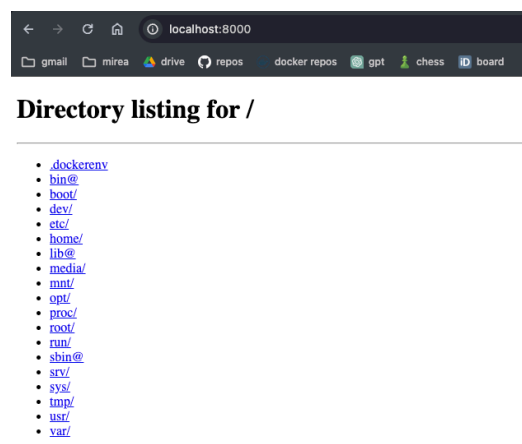


Рисунок 14 – Результат запуска python веб-сервера на порту 8000 с пробрасыванием

портов наружу

На рисунке 15 отображен процесс запуска python веб-сервера на порту 8000 с пробрасыванием портов наружу. При этом порт веб-сервера (8000) маппится на порт 8888 локальной машины

```
→ ~ docker run -it -p8888:8000 python python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.65.1 - - [23/Apr/2024 16:40:01] "GET / HTTP/1.1" 200 -
192.168.65.1 - - [23/Apr/2024 16:40:01] code 404, message File not found
192.168.65.1 - - [23/Apr/2024 16:40:01] "GET /favicon.ico HTTP/1.1" 404 -
```

Рисунок 15 – Запуск python веб-сервера на порту 8080 с пробрасыванием портов наружу (+ маппинг порта веб-сервера на 8888)

На рисунке 16 отображен результат запуска python веб-сервера на порту 8080 с пробрасыванием портов наружу

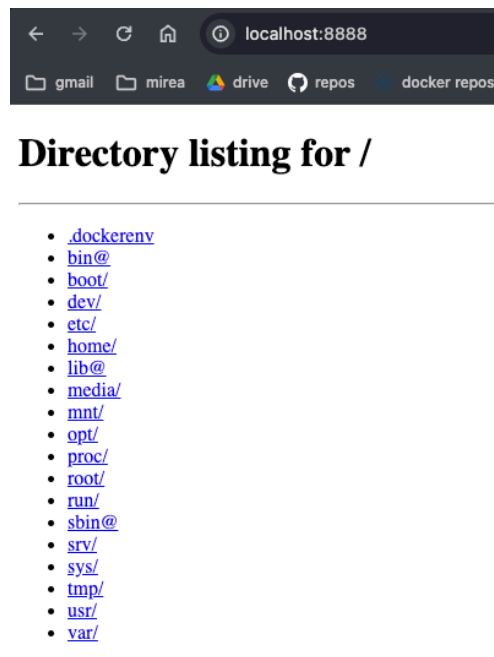


Рисунок 16 – Результат запуска python веб-сервера на порту 8080 с пробрасыванием портов наружу (+ маппинг порта веб-сервера на 8888)

4. Именованные контейнеры, остановка и удаление

На рисунке 17 отображен процесс завершения работы контейнера помощью CTRL + C


```
→ ~ docker run -it -p8888:8000 python python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.65.1 - - [23/Apr/2024 16:40:01] "GET / HTTP/1.1" 200 -
192.168.65.1 - - [23/Apr/2024 16:40:01] code 404, message File not found
192.168.65.1 - - [23/Apr/2024 16:40:01] "GET /favicon.ico HTTP/1.1" 404 -
^C
Keyboard interrupt received, exiting.
→ ~ █
```

Рисунок 17 – Завершение работы контейнера с помощью CTRL + C

На рисунке 18 отображен процесс запуска контейнера с detach-флагом. В этом случае запущенный контейнер не блокирует терминал.

```
→ ~ docker run -d -p8888:8000 python python -m http.server
dc31319295e5377074e3f95d67b94b268c3e9a9cb206a77654675f601ced5c15
→ ~ █
```

Рисунок 18 – Запуск контейнера с detach-флагом

На рисунке 19 отображен процесс запуска контейнера с задаванием имени контейнера

```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
→ ~ docker run -d --name server -p8888:8000 python python -m http.server
4fbadb9f4b427b7983eb204123d7686e506cb2eb93b302a37caabe26b0f010e4
→ ~ █
```

Рисунок 19 – Запуск контейнера с задаванием имени контейнера

На рисунке 20 отображен процесс просмотра всех контейнеров с фильтром pyserver

```
→ ~ docker ps | grep server
4fbadb9f4b42 python "python -m http.serv..." 44 seconds ago Up 44 seconds
0.0.0.0:8888->8000/tcp server
→ ~ █
```

Рисунок 20 – Запуск команды docker ps | grep server

На рисунке 21 отображен процесс просмотра логов контейнера с именем pyserver

```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
→ ~ docker logs server
192.168.65.1 - - [23/Apr/2024 16:46:04] "GET / HTTP/1.1" 200 -
→ ~ █
```

Рисунок 21 – Запуск команды docker logs

На рисунке 22 отображен процесс остановки контейнера server

```
→ ~ docker stop server
server
→ ~
```

Рисунок 21 – Запуск команды docker stop

На рисунке 23 отображен процесс запуска контейнера server после его остановки. Выдает ошибку, для решения конфликта необходимо удалить контейнер перед запуском.

```
→ ~ docker stop server
server
→ ~ docker run -d --name server -p8888:8000 python python -m http.server
docker: Error response from daemon: Conflict. The container name "/server" is al
ready in use by container "4fbadb9f4b427b7983eb204123d7686e506cb2eb93b302a37caab
e26b0f010e4". You have to remove (or rename) that container to be able to reuse
that name.
See 'docker run --help'.
→ ~
```

Рисунок 23 – Запуск контейнера server после его остановки

На рисунке 24 отображен процесс удаления и запуска контейнера. Теперь запуск происходит без ошибок

```
→ ~ docker rm server
server
→ ~ docker run -d --name server -p8888:8000 python python -m http.server
76a1739f3d8748a26f3c064ce77f0551073757c04b3c3f87cc88b90fea53f504
→ ~
```

Рисунок 24 – Процесс удаления и запуска контейнера

На рисунке 25 отображен процесс тестирования флага `--rm` – задает автоматическое удаление контейнера после завершения работы

```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
→ ~ docker run --rm --name server -p8888:8000 python python -m http.server
^CServing HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...

Keyboard interrupt received, exiting.
→ ~ docker stop server
Error response from daemon: No such container: server
→ ~
```

Рисунок 25 – Процесс тестирования флага `--rm`

5. Постоянное хранение данных

На рисунке 26 отображен запуск контейнера, который будет отдавать содержимое директории `/mnt`

```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
→ ~ docker run -p8000:8000 --name server --rm -d python python -m http.server -
d /mnt
3dfa9c1a1d3097271d31433646c1c723eceda0e4b328c7c5f9138762278a1dca
→ ~ █
```

Рисунок 26 – Запуск контейнера, который будет отдавать содержимое директории /mnt

На рисунке 28 отображен процесс записи hello world в hi.txt с помощью команды `cd mnt && echo "hello world" > hi.txt`

```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
→ ~ docker exec -it server bash
root@3dfa9c1a1d30:/# cd mnt && echo "hello world" > hi.txt
root@3dfa9c1a1d30:/mnt# exit
exit
→ ~ █
```

Рисунок 28 – Процесс записи hello world в hi.txt

На рисунке 29 отображен процесс захода на веб-сервер, отображена содержимое директории mnt, файл hi.txt

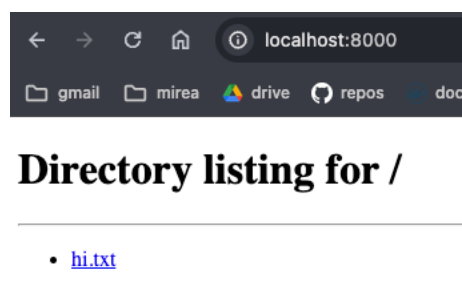


Рисунок 29 – Hi.txt файл на веб-сервере

На рисунке 30 отображен остановки контейнера и запуска заново

```
ryoichi — ryoichi@invoker — ~ — -zsh — 80x24
[→ ~ docker stop server
server
[→ ~ docker run -p8000:8000 --name server --rm -d python python -m http.serve]
r -d /mnt
bacbec6d96baec72dc48080187a377ac0ef4acc51a7b098b47d05e5a77c5a079
→ ~ █
```

Рисунок 30 – Остановка контейнера и запуск заново

На рисунке 31 отображен результат просмотра директории веб-сервера после перезапуска контейнера

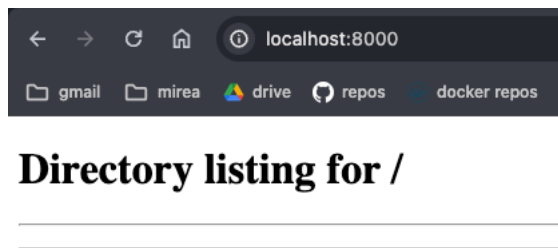


Рисунок 31 – Результат просмотра директории веб-сервера
а. Тома

На рисунке 32 отображен запуск контейнера с монтированным томом с помощью параметра -v и маппинга пути в ubuntu на путь на локальной машине

```
[→ ~ docker run -p8000:8000 --name server --rm -d -v /Users/ryoichi/Documents/se  
m4/trpp/pr4/mnt python python -m http.server -d /mnt  
3b6439ed1c3ba8942a956dec94ab4d0f4f93f9a017f7fd24e110be449d73effd  
→ ~ █
```

Рисунок 32 – Запуск контейнера с монтированным томом

На рисунке 33 отображено создание файла hi.txt в контейнере

```
ryoichi — docker exec -it server bash — docker — com.docker.cli ◀ docker...  
[root@3b6439ed1c3b:/# cd mnt && echo "hello world" > hi.txt  
root@3b6439ed1c3b:/mnt# █
```

Рисунок 33 – Создание файла hi.txt

На рисунке 34 отображен процесс остановки и еще одного запуска контейнера

```
[→ ~ docker stop server  
^R  
^R  
server  
[→ ~ docker run -p8000:8000 --name server --rm -d -v /Users/ryoichi/Documents/se  
m4/trpp/pr4/mnt python python -m http.server -d /mnt  
5b3b0a7bac89f2140599ab24183aea5b34778ed1b5c55f66c5742948e83ad656  
→ ~ █
```

Рисунок 34 – Перезапуск контейнера

На рисунке 35 отображена проверка на наличие файла hi.txt после перезапуска. Файл присутствует

```
ryoichi — docker exec -it server bash — docker — com.docker.cli ◀ docker...  
[→ ~ docker exec -it server bash  
[root@6b5c87027840:/# cd mnt  
[root@6b5c87027840:/mnt# ls  
hi.txt  
root@6b5c87027840:/mnt# █
```

Рисунок 35 – Проверка на наличие файла hi.txt

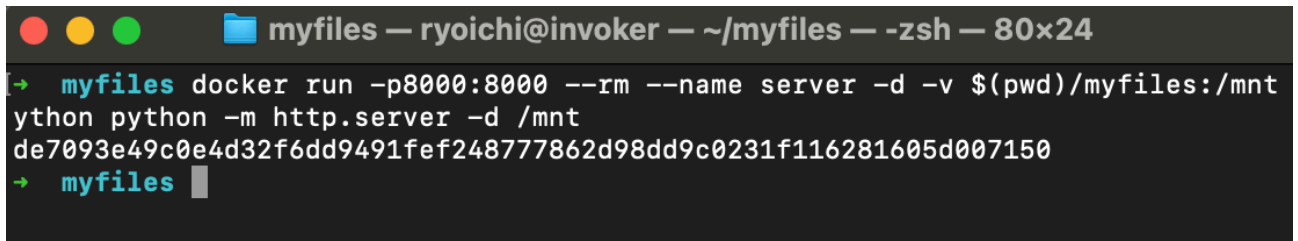
На рисунке 36 отображен просмотр места, где хранятся файлы с помощью команды `docker inspect server`,

```
"Mounts": [  
  {  
    "Type": "bind",  
    "Source": "/Users/ryoichi/Documents/sem4/trpp/pr4/mnt",  
    "Destination": "/mnt",  
    "Mode": "",  
    "RW": true,  
    "Propagation": "rprivate"  
  },  
],
```

Рисунок 36 – Просмотр места, где хранятся файлы

b. Монтирование директорий и файлов

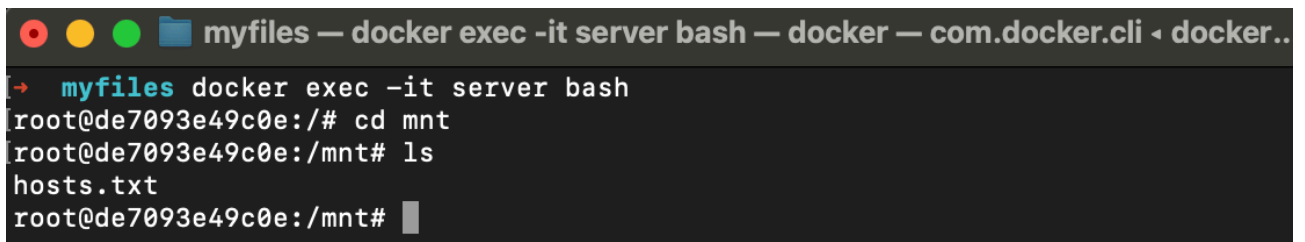
На рисунке 37 отображен запуск контейнера с пробрасыванием директории



```
myfiles — ryoichi@invoker — ~/myfile — zsh — 80x24  
→ myfiles docker run -p8000:8000 --rm --name server -d -v $(pwd)/myfile:/mnt  
python http.server -d /mnt  
de7093e49c0e4d32f6dd9491fef248777862d98dd9c0231f116281605d007150  
→ myfiles
```

Рисунок 37 – Запуск контейнера с пробрасыванием директории

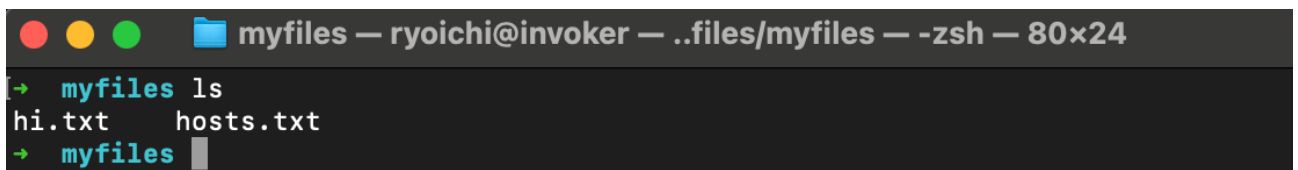
На рисунке 38 отображен процесс просмотра файлов проброшенной директории



```
myfiles — docker exec -it server bash — docker — com.docker.cli < docker..  
→ myfiles docker exec -it server bash  
root@de7093e49c0e:/# cd mnt  
root@de7093e49c0e:/mnt# ls  
hosts.txt  
root@de7093e49c0e:/mnt#
```

Рисунок 38 – Процесс просмотра файлов проброшенной директории

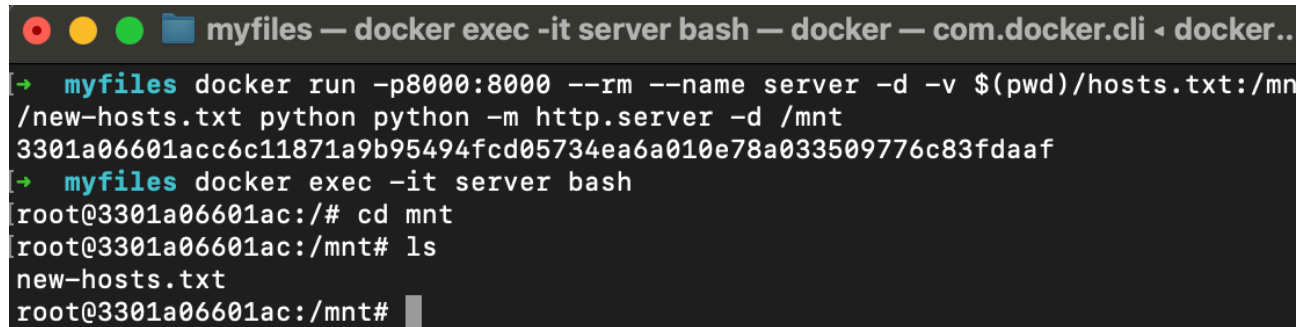
На рисунке 39 отображен процесс просмотра директории `myfile` на хостовой машине после создания файла `hi.txt` в контейнере



```
myfiles — ryoichi@invoker — ../myfile — zsh — 80x24  
→ myfiles ls  
hi.txt    hosts.txt  
→ myfiles
```

Рисунок 39 – Процесс просмотра директории `myfile` на хостовой машине после создания файла `hi.txt` в контейнере

На рисунке 40 отображен процесс монтирования одного файла со сменой имени

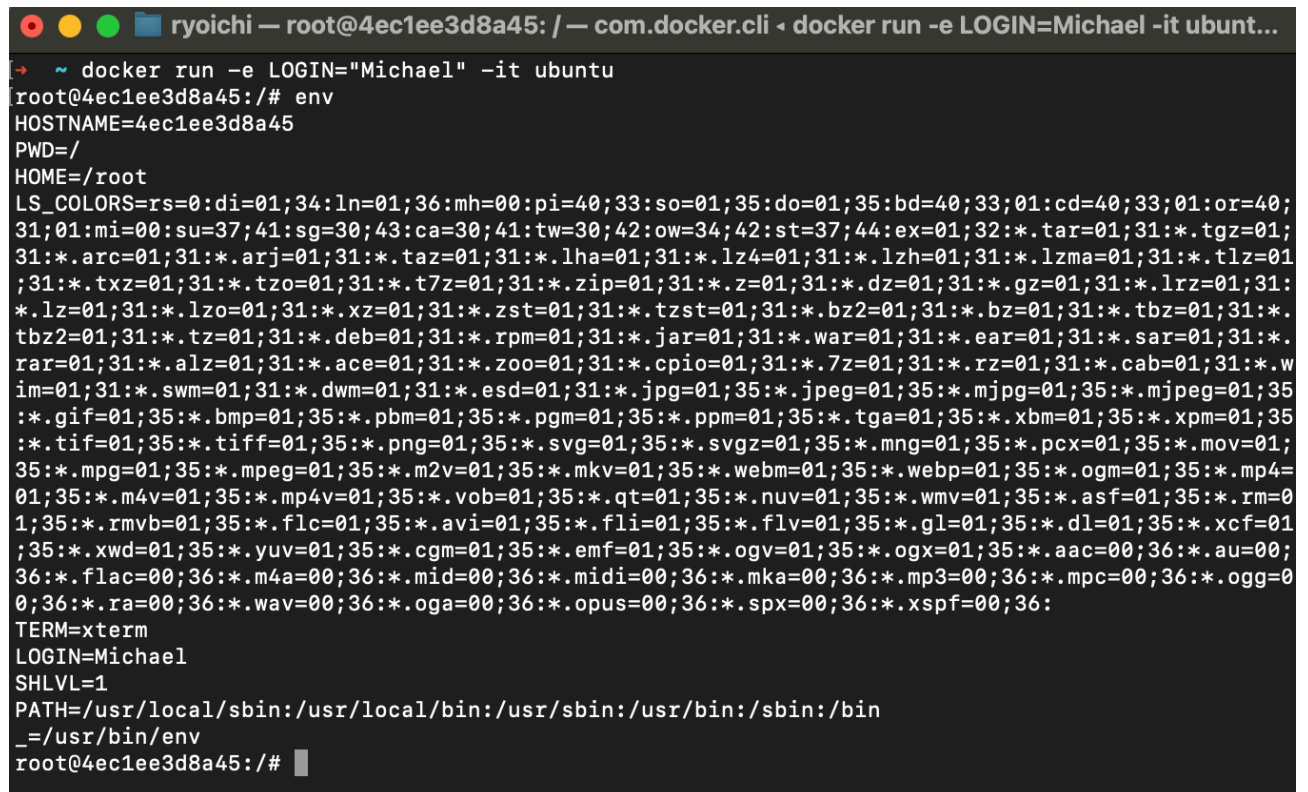


```
myfiles — docker exec -it server bash — docker — com.docker.cli < docker..
→ myfiles docker run -p8000:8000 --rm --name server -d -v $(pwd)/hosts.txt:/mnt
/new-hosts.txt python python -m http.server -d /mnt
3301a06601acc6c11871a9b95494fcd05734ea6a010e78a033509776c83fdaaf
→ myfiles docker exec -it server bash
root@3301a06601ac:/# cd mnt
root@3301a06601ac:/mnt# ls
new-hosts.txt
root@3301a06601ac:/mnt#
```

Рисунок 40 – Процесс монтирования одного файла со сменой имени

6. Переменные окружения

На рисунке 41 отображен процесс добавления переменной окружения LOGIN="Michael" и просмотра всех переменных контейнера с помощью терминала Bash



```
ryoichi — root@4ec1ee3d8a45: / — com.docker.cli < docker run -e LOGIN=Michael -it ubuntu...
→ ~ docker run -e LOGIN="Michael" -it ubuntu
root@4ec1ee3d8a45:/# env
HOSTNAME=4ec1ee3d8a45
PWD=/
HOME=/root
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;
31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;
31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01
;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:
*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.
tbz2=01;31:*.tzo=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.
rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.w
im=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35
:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35
:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;
35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.webp=01;35:*.ogm=01;35:*.mp4=
01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=0
1;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01
;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;
36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=0
0;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
TERM=xterm
LOGIN=Michael
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_=/usr/bin/env
root@4ec1ee3d8a45:/#
```

Рисунок 41 – Тестирование указывания переменных окружения при запуске контейнера

7. Dockerfile

На рисунках 42-43 отображены докер-файл и процесс билда образа из докер-файла


```
pr4 — nano Dockerfile — nano — nano Dockerfile — 96x28
UW PICO 5.09 File: Dockerfile Modified

FROM ubuntu:20.04
RUN apt update \
    && apt install -y python3 fortune \
    && cd /usr/bin \
    && ln -s pyhton3 pyhton
RUN /usr/games/fortune > /mnt/greeting-while-building.txt
ADD ./data /mnt/data
EXPOSE 80
CMD ["python", "-m", "http.server", "-d", "/mnt/", "80"]

^G Get Help ^O WriteOut ^R Read File ^Y Prev Pg ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where is ^V Next Pg ^U UnCut Text ^T To Spell
```

Рисунок 42 – Докер-файл

```
pr4 — ryoichi@invoker — ..sem4/trpp/pr4 — zsh — 96x28
→ pr4 git:(main) × mkdir data
→ pr4 git:(main) × docker build .
[+] Building 10.9s (9/9) FINISHED docker:desktop-linux
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 678B 0.0s
=> [internal] load metadata for docker.io/library/ubuntu:20.04 0.9s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/4] FROM docker.io/library/ubuntu:20.04@sha256:71b82b8e734f5cd0b3533a16f40ca1271f28 2.5s
=> => resolve docker.io/library/ubuntu:20.04@sha256:71b82b8e734f5cd0b3533a16f40ca1271f28 0.0s
=> => sha256:71b82b8e734f5cd0b3533a16f40ca1271f28d87343972bb4cd6bd6c38f1 1.13kB / 1.13kB 0.0s
=> => sha256:d40777b9a5329d3611f7bc40738b4cd459c8b1508fa8cc7c1555674c4067159 424B / 424B 0.0s
=> => sha256:8dcc83a51b0096c961e9431235462f6402e148368c77d99392a51a0da5b 2.31kB / 2.31kB 0.0s
=> => sha256:fb67143098b346a05ff35775af9ba34ccf9a89e4079f4ceb9a51b05ae 25.97MB / 25.97MB 1.7s
=> => extracting sha256:fb67143098b346a05ff35775af9ba34ccf9a89e4079f4ceb9a51b05ae257e78c 0.7s
=> [internal] load build context 0.0s
=> => transferring context: 26B 0.0s
=> [2/4] RUN apt update && apt install -y python3 fortune && cd /usr/bin && ln -s python 7.1s
=> [3/4] RUN /usr/games/fortune > /mnt/greeting-while-building.txt 0.2s
=> [4/4] ADD ./data /mnt/data 0.0s
=> exporting to image 0.2s
=> => exporting layers 0.2s
=> => writing image sha256:34350f770d8a769cd7d5b838c5dab10a412e0c9433f5df763be35a1e9ace9 0.0s

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
→ pr4 git:(main) ×
```

Рисунок 43 – Билд образа из докер-файла

На рисунке 44 отображены процесс запуска контейнера

```
pr4 — docker run --rm -it -p8099:80 343 — docker — com.docker.cli • docker run --rm -it -...
→ pr4 git:(main) ✖ docker run --rm -it -p8099:80 343
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Рисунок 44 – Запуск контейнера

На рисунке 45 отображены процесс тестирования запущенного контейнера

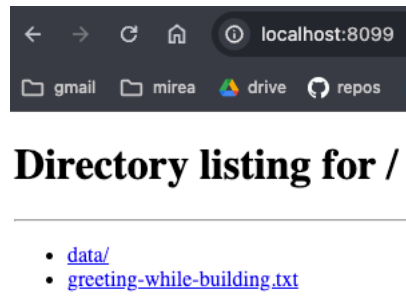


Рисунок 45 – Тестирование запущенного контейнера

8. Индивидуальные задания

На рисунке 46-47 отображен докер-файл и процесс билда образа из докер-файла

```
FROM ubuntu:20.04
RUN apt update \
&& apt install -y php-cli \
&& apt install -y python3 fortune \
&& cd /usr/bin \
&& ln -s python3 python
ADD ./data/student.txt /mnt/files/student.txt
EXPOSE 80
CMD ["python", "-m", "http.server", "-d", "/mnt/", "80"]
```

Рисунок 46 – Докер-файл


```
→ pr4 git:(main) × docker build .
[+] Building 16.0s (9/9) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile              0.0s
=> => transferring dockerfile: 689B                             0.0s
=> [internal] load metadata for docker.io/library/ubuntu:20.04  1.2s
=> [auth] library/ubuntu:pull token for registry-1.docker.io    0.0s
=> [internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                    0.0s
=> CACHED [1/3] FROM docker.io/library/ubuntu:20.04@sha256:71b82b8e734f5cd0b3533a16f40ca 0.0s
=> [2/3] RUN apt update && apt install -y php-cli && apt install -y python3 fortune && 14.5s
=> [internal] load build context                                0.0s
=> => transferring context: 155B                                 0.0s
=> [3/3] ADD ./data/student.txt /mnt/files/student.txt         0.0s
=> exporting to image                                           0.3s
=> => exporting layers                                          0.3s
=> => writing image sha256:25622e4ab2785cf07161ca67e56157f52c6e9518191d868891df78e883123 0.0s

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
→ pr4 git:(main) ×
```

Рисунок 47 – Докер-файл

На рисунке 48 отображены процесс запуска контейнера с образом из докер-файла

```
pr4 — docker run --rm -it -p8822:80 256 — docker — com.docker.cli < docker run --rm -it -...
[→ pr4 git:(main) × docker run --rm -it -p8822:80 256
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Рисунок 48 – Запуск контейнера с образом из докер-файла

На рисунке 49 отображены процесс тестирования запущенного контейнера с образом из докер-файла

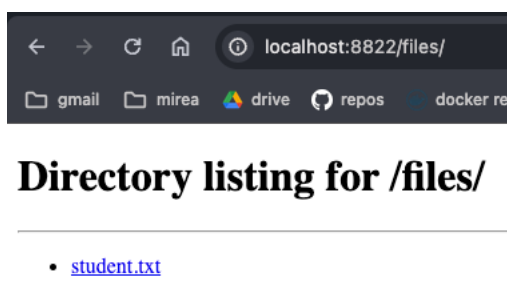


Рисунок 49 – Тестирование запущенного контейнера с образом из докер-файла

Вывод

В ходе работы мы познакомились с Docker, изучили его возможности, в том числе запуск контейнеров из образов, создание собственных образов, монтирование данных с хост-машины в контейнер, переменные окружения, работу с портами.