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 «Технологии разработки программных приложений»

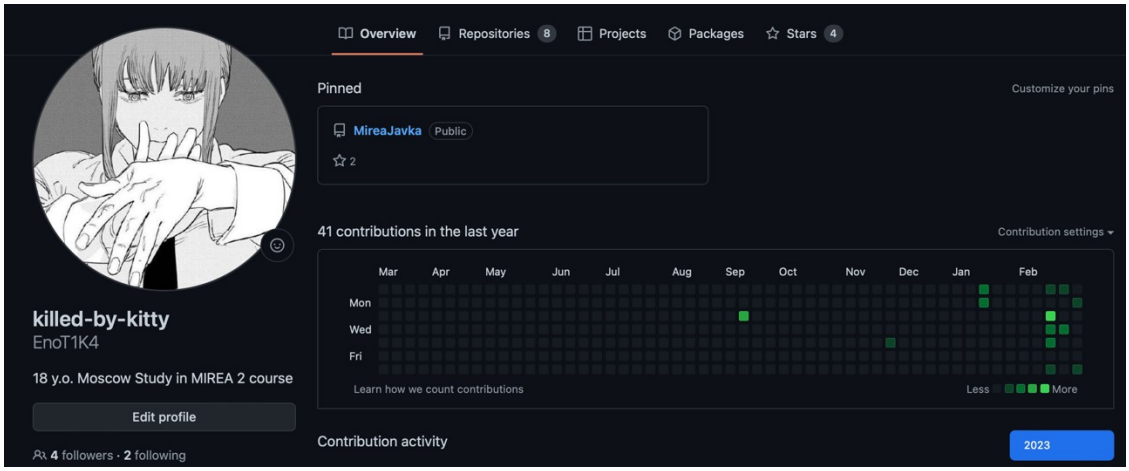
Тема практической работы: «Системы контроля версий»

Выполнил:
Студент группы **ИНБО-10-21** Дубровский В.Ф.

Проверил: кандидат технических наук, доцент
Жматов Д.В.

Москва 2023

Вариант 5



1 Образы

Посмотрите на имеющиеся образы: docker images.
Загрузите образ: docker pull ubuntu — будет загружен образ ubuntu:latest — последняя доступная

версия. Длязагрузкиконкретнойверсии,нужноуказатьтег,например,12.04:docker pull ubuntu:12.04. Посмотрите на имеющиеся образы ещё раз: docker images — должны появиться новые загруженные

образы. Посмотрите список контейнеров, выполнив команду: docker ps -a

2 Изоляция

Посмотрим информацию о хостовой системе, выполнив команду hostname. Выполните её ещё один раз. Вопрос: одинаковый ли результат получился при разных запусках?

Попробуемвыполнитьтожесамоевконтейнерах.Выполнитедваразакомандуdocker run ubuntu hostname. Вопрос: Одинаковый ли результат получился при разных запусках?

В случае запуска команды в контейнерах, ответ будет немного отличаться, будет разный hostname. Так происходит, потому что из одного образа ubuntu были запущены два изолированных контейнера, поэтому у них и был разный hostname.
Заново выполните docker ps -a — там должны появиться запущенные ранее контейнеры.

Запуск контейнеров производится командой:
docker run --флаги --докера имя_контейнера команда для запуска -и --флаги --запуска --программы.

Запустите bash в контейнере: docker run ubuntu bash. Ничего не произошло. Это не баг. Интерак- тивные оболочки выйдут после выполнения любых скриптовых команд, если только они не будут

запущены в интерактивном терминале — поэтому для того, чтобы этот пример не завершился, вам нужно добавить флаги -i -t или сгруппированно -it: docker run -it ubuntu bash.

Выполняя запуск контейнера, указывая образ ubuntu, неявно указывался образ ubuntu:latest. Сле- довательно, следующие команды равнозначны:

• docker run ubuntu hostname
• docker run ubuntu:latest hostname

Если бы мы хотели запустить ubuntu:12.04, то нужно было бы выполнить команду docker run ubuntu:12.04 hostname

3 Работа с портами

Для начала, загрузите 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. Завершите работу веб-сервера, нажав комбинацию клавиш Ctrl+C.

Для проброса портов используется флаг -p hostPort:containerPort

Добавьте его, чтобы пробросить порт 8000:
docker run -it -p8000:8000 python python -m http.server — теперь по адресу http://0.0.0.0: 8000/ (если не открывается на Windows, то вместо 0.0.0.0 нужно указать localhost) открывается содержимое корневой директории в контейнере.

Для того, чтобы доступный в контейнере на порту 8000 веб-сайт в хостовой системе открывался на порту 8888, необходимо указать флаг -p 8888:8000:
docker run -it -p8888:8000 python python -m http.server.

Завершите работу веб-сервера, нажав комбинацию клавиш Ctrl+C.

4 Именованные контейнеры, остановка и удаление

Запуститеконтейнер:docker run -it -p8000:8000 python python -m http.server.НажмитеCtrl+C — выполнение завершится. Для того, чтобы запустить контейнер в фоне, нужно добавить флаг -d/--detach. Также определим имя контейнеру, добавив флаг --name.

docker run -p8000:8000 --name pyserver -d python python -m http.server
Убедитесь, что контейнер всё ещё запущен: docker ps | grep pyserver — вывод команды не должен

быть пустым. Для просмотра логов контейнера, воспользуйтесь командой docker logs pyserver.

Для того, чтобы остановить выполнение контейнера, существует команда docker stop pyserver. Од- нако, если снова попробовать запустить командой

docker run -it -p8000:8000 --name pyserver -d python python -m http.server,товозникнетошиб- ка: контейнер с таким именем существует. Его нужно удалить docker rm pyserver.

Для остановки и удаления контейнера можно воспользоваться командой docker rm -f pyserver вме- сто выполнения двух отдельных команд stop и rm. После удаления контейнер с таким именем можно будет создать заново.

Для того, чтобы контейнер удалялся после завершения работы, нужно указать флаг --rm при его запуске — далее в работе мы будем использовать данный флаг:
docker run --rm -p8000:8000 --name pyserver -d python python -m http.server

5 Постоянное хранение данных

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.
Вопрос: Что значат остальные флаги запуска? Где здесь команда, которая выполнится в контейне- ре?

Длятого,чтобыпопастьвужезапущенныйконтейнер,существуеткомандаdocker 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.

Если открыть http://0.0.0.0:8000/, там будет доступен файл hi.txt. Остановим контейнер: docker stop pyserver, а затем снова запустим:

docker run -p8000:8000 --name pyserver --rm -d python python -m http.server -d /mnt.
Как мы видим, файл hi.txt пропал — это неудивительно, ведь мы запустили другой контейнер, а ста-

рый был удалён после завершения работы (флаг --rm). Остановим контейнер: docker stop pyserver. Для того, чтобы не терялись какие-то данные (например, если запущен контейнер с СУБД, то чтобы

не терялись данные из неё) существует механизм монтирования.

5.1 Тома

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

Попробуйте снова создать контейнер, но уже с примонтированным томом:

docker run -p8000:8000 --rm --name pyserver -d \
-v $(pwd)/myfiles:/mnt python python -m http.server -d /mnt

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.

Для управления томами существует команда docker volume, ознакомиться с которой предлагается самостоятельно.

5.2 Монтирование директорий и файлов

Сперва, остановите контейнер, созданный на предыдущем шаге: docker stop pyserver.
Иногда требуется пробросить в контейнер конфигурационный файл или отдельную директорию. Для

этого используется монтирование директорий и файлов.
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

Запустте контейнер:

```
docker run -p8000:8000 --rm --name pyserver -d -v $(pwd)/myfiles:/mnt python \
  python -m http.server -d /mnt
```

Команда pwd — выведет текущую директорию, например: /home/user/dome-directory, в итоге полу- чился абсолютныйпуть до файла: /home/user/dome-directory/myfiles.
Обратный слеш (\) перед переводом строки экранирует символ перевода строки и позволяет написать одну команду в несколько строк.

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

Создайте файл echo "hello world" > hi.txt, а затем выйдите из контейнера: exit. Теперь на хостовой машине в директории myfiles/ появится файл hi.txt. Проверить можно командой ls myfiles.

Остановите контейнер: docker stop pyserver.
Для того, чтобы примонтировать один файл, нужно указать ключ -v, например:

-v $(pwd)/myfiles/host.txt:/mnt/new-name-of-host.txt – файлу в контейнере присвоится другое имя: new-name-of-host.txt.

Если на Windows возникают ошибки при монтировании, убедитесь, что вы используете bash , а не cmd.exe.

6 Переменные окружения

Для передачи переменных окружения внутрь контейнера используется ключ -e. Например, чтобы передать в контейнер переменную окружения MIREA во 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. Среди списка переменных будет и MIREA

7 Dockerfile

Соберите образ, в который будут установлены дополнительные пакеты, примонтируйте директорию и установите команду запуска. Для этого создаётся файл Dockerfile (без расширения).

1. 1 FROM
2. 2 RUN

3 4

ubuntu :20.04
apt update \
&& apt install -y python3 fortune \
&& cd /usr/bin \
&& ln -s python3 python
/usr/games/fortune > /mnt/greeting-while-building.txt ./data /mnt/data

5

6. 6 RUN
7. 7 ADD
8. 8 EXPOSE 80
9. 9 CMD ["python", "-m", "http.server", "-d", "/mnt/", "80"]

Будьте внимательны при копировании, могут скопироваться неправильные минусы и лишние про- белы.
В строке (1) указывается базовый образ, на основе которого будет строиться новый образ.
В строках (2-5) указана команда, которая выполнится в процессе сборки. На самом деле, там выпол- няются несколько команд, соединённых && для того, чтобы создавать меньше слоёв в образе.

В строках (6) тоже указана команда, которая сгенерирует случайную цитату и перенаправит вывод в файл /mnt/greeting-while-building.txt. Файл будет сгенерирован во время сборки образа.
В строке (7) копируется всё содержимое директории ./data хостовой машины в директорию /mnt, которая будет доступна в контейнере.

В строке (8) указывается, какой порт у контейнера будет открыт.
В строке (9) указывается команда, которая будет выполнена при запуске, где 80 — порт, который будет слушать веб-сервер.

Соберите образ с тегом mycoolimage с помощью команды docker build -t mycoolimage . Точка в конце указывает на текущую директорию, где лежит Dockerfile.

Запуск производится командой docker run --rm -it -p8099:80 mycoolimage, где порт 8099 — порт на хостовой машине.

```
Last login: Thu Mar  9 19:34:18 on console
[dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
[dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
d0a4bfa485d1: Downloading  15.94MB/27.35MB
d0a4bfa485d1: Pull complete
Digest: sha256:2adf22367284330af9f832ffefb717c78239f6251d9d0f58de50b86229ed1427
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ %
[dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker images
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
ubuntu       latest    730eeb702b69   9 days ago   69.2MB
[dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker ps-a
docker: 'ps-a' is not a docker command.
See 'docker --help'
[dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Рисунок 1 – задание 1

```
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % hostname
MacBook-Pro-Dubrovskij.local
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker run ubuntu hostname
cde5cbfcae19
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker ps -a
CONTAINER ID   IMAGE     COMMAND      CREATED         STATUS
cde5cbfcae19   ubuntu    "hostname"   9 seconds ago   Exited (0) 8 seconds ago
oodall
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker run ubuntu bash
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % -it: docker run -it ubuntu bash
zsh: command not found: -it:
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker run -it ubuntu bash
root@36b3f0e0e0dd:/# docker pull python
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker pull python
Using default tag: latest
latest: Pulling from library/python
0f5fe16b1836: Pull complete
7b8a090f23f2: Pull complete
ec29ff8525a3: Pull complete
8747d4a988af: Pull complete
fb701404a7a0: Downloading  188.6MB/189.7MB
fb701404a7a0: Extracting  189.7MB/189.7MB
fb701404a7a0: Pull complete
e095422dfe12: Pull complete
094d52cee9a2: Pull complete
702c50ac6862: Pull complete
bf030d16ee42: Pull complete
Digest: sha256:d3c16df33787f3d03b2e096037f6deb3c1c5fc92c57994a7d6f2de018de01a
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
```

Рисунок 2 – задание 2

```
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker run -it python python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
^C
Keyboard interrupt received, exiting.
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker run -it -p8000:8000 python python -m htt
er
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
172.17.0.1 - - [10/Mar/2023 12:22:44] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [10/Mar/2023 12:22:44] code 404, message File not found
172.17.0.1 - - [10/Mar/2023 12:22:44] "GET /favicon.ico HTTP/1.1" 404 -
```

Directory listing for /

- .dockerenv
- bin/
- boot/
- dev/
- etc/
- home/
- lib/
- media/
- mnt/
- opt/
- proc/
- root/
- run/
- sbin/
- srv/
- sys/
- tmp/
- usr/
- var/

Рисунок 3 – задание 3

```
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker ps | grep pyserver
76d9bf629866   python    "python -m http.serv…"   About a minute ago   Up About a minute   0.0.0.0:8
000->8000/tcp   pyserver
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker logs pyserver
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker stop pyserver
pyserver
```

Рисунок 4 – задание 4

```
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker run --rm -p8000:8000 --name pyserver -d python
 python -m http.server
999d09b53e9d62e2aa6957241dfcd5d84f349c8bde90d76a9416022279b0b878
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker stop pyserver
pyserver
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker run -p8000:8000 --name pyserver --rm -d python
 python -m http.server -d /mnt
d672c04ca8260fef559aa3c234b36ff83ac5f8a094d7539d393a94e0748b3f77
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker exec -it pyserver bash
```

```
root@d672c04ca826:/# cd mnt && echo "hello world" > hi.txt
root@d672c04ca826:/mnt#
exit
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker stop pyserver

pyserver
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ %
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker run -p8000:8000 --name pyserver --rm -d python
 python -m http.server -d /mnt.
e113b2144bd35de9cf5db8dbdc1ebf7a5a7b2d94b594d03725db309366afff9e
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker stop pyserver
pyserver
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker run -p8000:8000 --rm --name pyserver -d \
-v $(pwd)/myfiles:/mnt python python -m http.server -d /mnt
67b9b592257cfb285410ae2337d3b0c7a7495829684e122999cfa02803602b9b
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker exec -it pyserver bash
root@67b9b592257c:/# cd mnt && echo "hello" > hi.txt
root@67b9b592257c:/mnt#
exit
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % docker inspect -f "{{json .Mounts }}" pyserver
[{"Type":"bind","Source":"/Users/dubrovskijvladislav/myfiles","Destination":"/mnt","Mode":"","RW":tr
ue,"Propagation":"rprivate"}]
```

Рисунок 6 – задание 5

```
   python -m http.server -d /mnt
ff27361a76d0313d8b0999d851f504b4843fcaa53d5b31f65973db072d6308d6
dubrovskijvladislav@MacBook-Pro-Dubrovskij myfiles % docker exec -it pyserver bash
root@ff27361a76d0:/# cd /mnt
root@ff27361a76d0:/mnt# ls
root@ff27361a76d0:/mnt# echo "hello world" > hi.txt
root@ff27361a76d0:/mnt# exit
exit
dubrovskijvladislav@MacBook-Pro-Dubrovskij myfiles % ls myfiles
hi.txt
dubrovskijvladislav@MacBook-Pro-Dubrovskij myfiles % docker stop pyserver
pyserver
dubrovskijvladislav@MacBook-Pro-Dubrovskij myfiles % touch Dockerfile
dubrovskijvladislav@MacBook-Pro-Dubrovskij myfiles % docker build -t mycoolimage .
[+] Building 1.1s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                0.0s
 => => transferring dockerfile: 549B                                                0.0s
 => [internal] load .dockerignore                                                   0.0s
 => => transferring context: 2B                                                     0.0s
 => [internal] load metadata for docker.io/library/ubuntu:20.04                     1.0s
 => [internal] load build context                                                   0.0s
 => => transferring context: 136B                                                   0.0s
 => [1/4] FROM docker.io/library/ubuntu:20.04@sha256:9fa30fcef427e5e88c76bc41ad37b7cc573e1d79  0.0s
 => CACHED [2/4] RUN apt update  && apt install -y python3 fortune  && cd /usr/bin  && ln -s  0.0s
 => CACHED [3/4] RUN /usr/games/fortune > /mnt/greeting-while-building.txt          0.0s
 => CACHED [4/4] ADD ./myfiles /mnt/data                                            0.0s
 => exporting to image                                                              0.0s
 => => exporting layers                                                             0.0s
 => => writing image sha256:3b30a2155ce16d93e43798719cab9350ce3247c5dd4a1b657206cd2556958d38  0.0s
 => => naming to docker.io/library/mycoolimage                                      0.0s
dubrovskijvladislav@MacBook-Pro-Dubrovskij myfiles % docker run --rm -it -p8099:80 mycoolimage
```

Рисунок 7 – задание 7

```
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % mkdir data
dubrovskijvladislav@MacBook-Pro-Dubrovskij ~ % cd data
dubrovskijvladislav@MacBook-Pro-Dubrovskij data % touch student.txt
dubrovskijvladislav@MacBook-Pro-Dubrovskij data % nano student.txt
```

Dockerfile

```
FROM ubuntu:20.10
RUN  apt install -y git
ADD ./data ./mnt/data
EXPOSE 25
CMD ["git", "-m", "http.server", "-d", "/mnt/", "25"]
```

Рисунки 8-9 – задание 8

Вывод

В ходе выполнения данной работы научился управлять системой сборки Docker.