Лабораторна робота №3

1. Python

1. З самого початку я намагався запустити проект, щоб подивитися взагалі що це таке, і яка там структура. По волі Божій, в мене вийшло встановити всі залежності, налаштувати проект та запустити venv (попри одрук в рідмі файлі сетапа для цієї частини лабки). `. ./.venv/bin/acticate`
Звісно, в процесі я постикався з помилками, потім в мене кудись зникла розмітка. Але, після ретельної перевірки та чергового запуску, все запрацювало і я зміг потикати проект на локальному хості.

Далі я пробував запустити білд і отримував помилки. Щільно подумавши, я зрозумів, що в мене не встановлений докер :D. Після інсталяції докера з офіційного сайту та авторизації, я згенерував залежності через pip freeze та перенаправив в текстовий файл.
“pip freeze > requirements.txt"

Потім я зробив опис Dockerfile. З ним мені допоміг GTP4 та гугл.

Нарешті зібрав перший образ:

```
PS C:\Users\Anton\PycharmProjects\mlab3\python> docker build -t firsttry .
```

Час збірки становив 12 секунд, а розмір 196 мегабайтів

```
PS C:\Users\Anton\PycharmProjects\mlab3\python> docker build -t firsttry .
[+] Building 12.0s (11/11) FINISHED
```

```
PS C:\Users\Anton\PycharmProjects\mlab3\python> docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
firsttry     latest    ab90c712f8b3   16 minutes ago   196MB
PS C:\Users\Anton\PycharmProjects\mlab3\python>
```

2. Далі зробив зміни в розмітці build/index.html, та додав коментарі.
Забілдив знову

```
PS C:\Users\Anton\PycharmProjects\mlab3\python> docker build -t secondtry .
[+] Building 1.8s (11/11) FINISHED
```

```
secondtry    latest    16462b3f1e82   38 seconds ago   196MB
```

Забілдилося все швидко, бо, виявляється, мені вдалося написати одразу оптимізований докер-файл, і він кешує кроки, тому білд пройшов дуже швидко.

3. Переписав докер-файл таким чином, щоб в теорії воно все було неоптимізоване.

Забілдив перший раз:

```
PS C:\Users\Anton\PycharmProjects\mlab3\python> docker build -t unoptbefore .
[+] Building 10.1s (11/11) FINISHED
```

Зробив зміни в розмітці та накидав коментарів в spaceship/app.py

Спробував збілдити знову

```
PS C:\Users\Anton\PycharmProjects\mlab3\python> docker build -t unoptafter .
[+] Building 11.4s (10/10) FINISHED
```

Як бачимо, тепер воно не кешується, і білдиться наново, ще й навіть довше ніж до цього.

Щодо розмірів, то вони однакові до змін, і після (216мб):

```
unoptafter          latest    c8a237c15981   About a minute ago   216MB
unoptbefore         latest    9a0eaabc7a90   3 minutes ago        216MB
```

4. Далі я змінив докерфайл (той шо оптимізований) так, щоб було взято за основу менший базовий образ (замість python:3.9-slim, що на дебіані, взяв python:3.10-alpine, що на базі alpine)

Збілдився за 6.8 секунд, доволі швидко.

```
[+] Building 6.8s (10/10) FINISHED                                  docker:default
```

А розмір зменшився до 121 мегабайта.

```
alpine        latest    2ba72afda66c   6 seconds ago       121MB
```

Тобто, при зміні базового образу на менший, білд йде швидше та розмір фінального образу менший, ніж був, що логічно.

5. Далі по завданню я додав у проект залежність numpy та ендпоінт, який генерує 2 випадкові матриці 10 на 10 та множить їх.

Спочатку збілдив новий образ на основі alpine (python:3.10-alpine)

Збілдилося за 12.3 секунди

```
PS C:\Users\Anton\PycharmProjects\mlab3\python> docker build -t alpinematrix .
[+] Building 12.3s (11/11) FINISHED
```

Розмір становив 264 мегабайти

```
alpinematrix   latest    57113eac1f7a   4 minutes ago    264MB
```

Потім на основі debian (python:3.9-slim)

Збілдилося за 10.3 секунди

```
PS C:\Users\Anton\PycharmProjects\mlab3\python> docker build -t slim39matrix .
[+] Building 10.3s (10/10) FINISHED
```

Розмір становив 340 мегабайт

```
slim39matrix   latest    4fa9559cdf29   10 seconds ago   340MB
```

Отже, на основі debian вийшло трохи швидше, але зайняло більше місця, порівняно з alpine. Різниця в часі не суттєва, а от різниця в розмірах вже більш відчувається. Також бачимо, що розмір і там, і там більший за той, що був в образа до встановлення нової залежності numpy та ендпоінта, тобто це теж привело до збільшення розміру образу десь на 124 мегабайти.

Примітка: в останній момент помітив, що замість 8080 завжди писав (в докер файлі) 80 порт, і тестив навіть теж на 80 (видно по комітам), не помічаючи, але останнім комітом закинув правильний порт, будь ласка не бийте.

2. GoLang

1. Спочатку я скопіював шаблон стартового репо по посиланню, потім почав виконувати команди з рідмі, і зіштовхнувся з проблемою, що не запускається сторінка локально. Через хвилин 10 второпав, що це ж вінда, і тут треба прописувати в кінці файлу розширення, якщо файл виконуваний, тобто замість “go build -o build/fizzbuzz” треба було просто написати go “build -o build/fizzbuzz.exe”, - і вуаля, все працює.

Далі я зробив dockerfile і почав білдити за допомогою знайомої команди

```
docker build -t fizzbuzz1try .
```

Це зайняло 26.4 секунди.

Та 935 мегабайтів.

Потім я хотів перевірити контейнер, але замість повідомлення, що все прослуховається і все нормально, мені виводилася інструкція, знову і знову

```
PS C:\Users\Anton\GolandProjects\mlab3\golang> docker run -p 8080:8080  --rm fizzbuzz1try
Usage:
  fizzbuzz [command]

Available Commands:
  completion  Generate the autocompletion script for the specified shell
  help        Help about any command
  query       Query if the number is fizz/buzz or fizzbuzz
  serve       Run an http server to anser fizzbuzz queries

Flags:
  -h, --help   help for fizzbuzz

Use "fizzbuzz [command] --help" for more information about a command.
```

Посидівши трохи над цим, я второпав, що треба в докерфайл додати ще “serve”. Я знову збілдив це все діло:

```
PS C:\Users\Anton\GolandProjects\mlab3\golang> docker build -t 1task .
[+] Building 2.0s (13/13) FINISHED
```

Дві секунди, бо кешування та оптимізований докер-файл, так би воно зайняло знову до 30 секунд. На цей раз все спрацювало, я зміг запустити контейнер.(на цей раз порт не переплутав, запускав та тестив на 8080 порті)

```
PS C:\Users\Anton\GolandProjects\mlab3\golang> docker run -p 8080:8080  --rm 1task
Listening on http://0.0.0.0:8080
```

Далі треба було перевірити, чи всі файли в контейнері потрібні для запуску проекту. Я не знав, яку команду використати, тому спитав у своїх найліпших друзів пораду – в гпт та гугла. Відповідь вбила:

```
PS C:\Users\Anton\GolandProjects\mlab3\golang> docker run -it --rm 1task /bin/sh
```

Що ж, на свій страх та ризик я використав цю команду, і вона… Спрацювала!

За допомогою всім відомої ls я вивів зміст контейнера:

```
/app # ls
Dockerfile  README.rst  build      cmd        fizzbuzz    go.mod     go.sum     lib        main.go    templates
```

Тут є всі потрібні для проекту файли, але також є і зайві, типу як Dockerfile, README, які теж туди скопіювалися.

2. Далі багатоетапна збірка.

Я з горем навпіл, після мільйону помилок, добився від гпт (та погугливши) потрібного коду для докер-файлу, і воно працює.

Я збілдив його

```
PS C:\Users\Anton\GolandProjects\mlab3\golang> docker build -t 2task .
[+] Building 5.8s (14/14) FINISHED
```

5 секунд, бо це не перша спроба, і воно докидувало зміни, грубо кажучи.

```
2task        latest    aa09e53bcc7c   24 minutes ago   8.97MB
```

І на мій подив, образ займає всього майже 8.97 мегабайтів, порівняно з першим завданням – це небо та земля, настільки маленький розмір, вау!

І найголовніше – працює ж!

```
PS C:\Users\Anton\GolandProjects\mlab3\golang> docker run -p 8080:8080 --rm 2task
Listening on http://0.0.0.0:8080
```

Залізти всередину, на жаль, не можна, бо наш скретч порожній. Тобто ми звільняємо купу місця, але жертвуємо нашим комфортом, а саме довгими танцями з бубнами, щоб налаштувати це все та неможливістю заглянути всередину.

3. Тепер замість scratch використовуємо образи із проекту distroless

Аналогічно зробив докер файл з distroless, тобто переробив той шо був вище.

Збілдив, дуже швидко, класно, комплімент моєму компу, мабуть

```
PS C:\Users\Anton\GolandProjects\mlab3\golang> docker build -t 3task .
[+] Building 4.9s (17/17) FINISHED
```

Розмір більший аж на 20 мегабайтів або лише на 20 мегабайтів, хм

```
3task         latest    65b223e79069   24 seconds ago   28.2MB
```

Проте знову працює, і це чудово

```
PS C:\Users\Anton\GolandProjects\mlab3\golang> docker run -p 8080:8080 --rm 3task
Listening on http://0.0.0.0:8080
```

Аналогічно не можна подивитися внутряк, на жаль, бо знову образ по-факту порожній.

Тобто, коротко кажучи, хоч багатоетапна збірка складна в налаштуванні та туди не можна залізти, вона економить нам купу місце, а це важливий аспект.

І взагалі go доволі некрасива оку мова, а налаштування було ще більш запарним (помилка на помилці, і зверху ще варнінги), не думав, що колись це скажу, але навіть пітон мені більше сподобався.

3. Мова на мій вибір (nodeJS)

Я обрав саме джаваскрипт, бо всі проєкти я переважно на ньому писав, він зручний та легкий. Як фреймворк я обрав “polka”, бо назва прикольна і він дуже легкий в налаштуванні. Потім написав простий веб-застосунок, який виводить “Hello world”, коли ми заходимо на http://127.0.0.1:8080/. Закинув всі потрібні файли та протестив локально, все норм, все працює, все супер.

Далі я написав докерфайл, гугл мені порекомендував спробувати node:14-alpine, я так і зробив, бо все одно потім буду тестити і інші основи.

Все запрацювало з першої спроби, забілдилося за 10.4 секунди, нічого поки дивного.

```
PS C:\Users\Anton\WebstormProjects\mlab3\js> docker build -t js1try .
[+] Building 10.4s (11/11) FINISHED
```

Щодо розміру, то 120 мегабайтів, небагато, але ж у нас і весь проект взагалі маленький по масштабам.

```
js1try       latest    deb7b4fb9ac5   3 minutes ago   120MB
```

Далі я спробував просто node:14 (цей образ на основі дебіана)

```
PS C:\Users\Anton\WebstormProjects\mlab3\js> docker build -t js2try .
[+] Building 20.3s (11/11) FINISHED
```

Білдилося довше, можна вже сказати, що тут більше буде розмір

```
js2try       latest    ef52286b4001   About a minute ago   912MB
```

Так і є, розмір катастрофічно великий, порівняно з тим, що був на alpine.

Погугливши, вияснив, що цей образ містить більше вбудованих інструментів та бібліотек, що робить його більшим за розміром порівняно з alpine версією.

Далі я вирішив взяти свіжішу версію ноди за основу (20) та slim, яка теж заснована на дебіані.

Збілдилося дуже швидко, всього 5.4 секунди

```
PS C:\Users\Anton\WebstormProjects\mlab3\js> docker build -t js3try .
[+] Building 5.4s (11/11) FINISHED
```

Розмір більший за той, що був у нас на alpine, але значно менший за node:14

```
js3try       latest    f1fbd126cfcc   About a minute ago   201MB
```

Тобто node:14-alpine у нас топ 1 по розмірам, там вийшло найкомпактніше.

Але в репо я залишив node:20-slim, бо мені так більше візуально подобається.

Далі тестимо багатоетапну збірку.

В мене з'явилися ідея потестити і alpine, і slim (20) для двоетапної збірки.

Почнемо з node:20-alpine

Якось зміг написати докерфайл для двоетапної збірки, зіблдилося за 16 секунд

```
PS C:\Users\Anton\WebstormProjects\mlab3\js> docker build -t 2stepsalpine .
[+] Building 16.8s (15/15) FINISHED
```

Розмір, як завжди, приємно дивує, всього 135 мегабайтів.

```
2stepsalpine   latest    e943dbd28814   20 seconds ago   135MB
```

Далі потестив, все працює (на диво).

Тепер спробуємо node:20-slim

Забілдилося дуже швидко, бо воно закешувало кроки, за секунду вже є.

```
PS C:\Users\Anton\WebstormProjects\mlab3\js> docker build -t 2stepslim .
[+] Building 1.4s (15/15) FINISHED
```

Розмір теж невеликий, 201мб , але більше, ніж з alpine.

```
2stepslim      latest    6670ee449c10   7 seconds ago    201MB
```

Отже, глобально ситуація аналогічно як було в минулій частині з golang: багатоетапна збірка займає мало місця, але складна і довго в написанні, хоча навіть без неї вийшло забілдити в 120 мегабайтів (alpine допоміг). Але загалом, якщо треба поменше місця, то стовідсотково можна використовувати багатоетапну збірку, бо в звичайній можна і позначку в гігабайт пробити, якщо якусь основу обрати не ту. Ну і не слід забувати, що при зміні базового образу на менший (більше стосується звичайної збірки), розмір фінального образу буде менший. Тобто якщо точно знаєш який образ за основу хочеш брати, то можна не паритись з налаштуванням багатоетапної збірки. Але все залежить від того, які інструменти потрібні, розмір, версія і так далі.

4. Висновки (по лабі)

Отже, якщо коротко, без води кажучи, то лабораторна робота мені максимально сподобалося, було цікаво тикать і розбиратися з докером. До цього про докер я тільки чув, тому прочитавши завдання було страшнувато, але походу завдання ставало все цікавіше і легше, навіть попри помилки та труднощі в процесі. Найголовніше, що зрозумів як це відбувається та який саме практичний і реальний зміст оцих всіх танців з докером.

Github repo