1. Python

1) Встановив всі необхідні пакети за допомогою команди **pip install -r .\requirements\backend.in**. Закріпив список залежностей за допомогою команди **pip freeze** (перенаправивши вивід в .\requirements\requirements.txt).
Створив **Dockerfile** для збірки образу. За допомогою команди **docker build -t lab3-python:v01 .** зібрав образ (знаходячись в каталозі, де сам застосунок та Dockerfile). Збірка зайняла 107.7 секунд (разом із завантаженням базового образу):

```
maksym@Ubuntu:~/Python$ sudo docker build -t lab3-python:v01 .
[+] Building 107.7s (9/9) FINISHED
```

Розмір образу склав 1.01 гігабайт:

```
maksym@Ubuntu:~/Python$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
lab3-python   v01       4a6bad1bb34e   3 minutes ago    1.01GB
```

2) Вніс зміни у файл build/index.html та повторно зібрав образ використавши команду **docker build -t lab3-python:v02 .** .
Цього разу час збірки склав 23.2 секунди, це зумовлено тим, що базовий образ **python:3.12.3-bullseye** (завантаження якого зайняло найбільше часу при першій збірці) вже був завантажений, тому не потребував повторного завантаження. Також другий пункт (WORKDIR /app) закешувався. Але інші два пункти, в тому числі і встановлення всіх залежностей, були виконані повторно.

```
maksym@Ubuntu:~/Python$ sudo docker build -t lab3-python:v02 .
[+] Building 23.2s (9/9) FINISHED                                docker:default
 => [internal] load build definition from Dockerfile                      0.0s
 => => transferring dockerfile: 225B                                      0.0s
 => [internal] load metadata for docker.io/library/python:3.12.3-bullseye  1.5s
 => [internal] load .dockerignore                                         0.0s
 => => transferring context: 2B                                           0.0s
 => [1/4] FROM docker.io/library/python:3.12.3-bullseye@sha256:dea6ffe7e8  0.0s
 => [internal] load build context                                         0.0s
 => => transferring context: 1.08kB                                       0.0s
 => CACHED [2/4] WORKDIR /app                                             0.0s
 => [3/4] COPY . /app                                                     0.1s
 => [4/4] RUN pip install -r requirements/requirements.txt               20.6s
 => exporting to image                                                    0.9s
 => => exporting layers                                                   0.8s
 => => writing image sha256:17377e13d334d716037c932e1c217c98e7c25c24a060c  0.0s
 => => naming to docker.io/library/lab3-python:v02                        0.0s
```

Розмір образу залишився незмінним (1.01 гігабайт)
3) Вніс зміни в Dockerfile для забезпечення кешування та зібрав образ заново. Потім знову вніс зміни в build/index.html та зібрав образ:

```
maksym@Ubuntu:~/Python$ sudo docker build -t lab3-python:v03 .
[+] Building 1.0s (10/10) FINISHED                                docker:default
 => [internal] load build definition from Dockerfile                        0.0s
 => => transferring dockerfile: 255B                                        0.0s
 => [internal] load metadata for docker.io/library/python:3.12.3-bullseye   0.6s
 => [internal] load .dockerignore                                           0.0s
 => => transferring context: 2B                                             0.0s
 => [1/5] FROM docker.io/library/python:3.12.3-bullseye@sha256:dea6ffe7e8   0.0s
 => [internal] load build context                                           0.0s
 => => transferring context: 1.08kB                                         0.0s
 => CACHED [2/5] WORKDIR /app                                               0.0s
 => CACHED [3/5] COPY requirements/requirements.txt /app                    0.0s
 => CACHED [4/5] RUN pip install -r requirements.txt                        0.0s
 => [5/5] COPY . /app                                                       0.1s
 => exporting to image                                                      0.1s
 => => exporting layers                                                     0.1s
 => => writing image sha256:742b2fa37cb5524a0cdc609a04347301220c51bd69197   0.0s
 => => naming to docker.io/library/lab3-python:v03                          0.0s
```

Як видно, що час збірки став значно менший (1 секунда) за рахунок кешування. Розмір образу знову незмінний (1.01 гігабайт).

4) Змінив в Dockerfile базовий образ з python:3.12.3-bullseye на python:3.12.3-alpine. Після збірки образу видно, що час збірки більше ніж в два рази менше за той, який був потрібний для збирання першої версії, а саме 48.6 секунди цього разу в порівнянні з 107.7 секундами збирання минулого разу (але також варто зауважити, що цього разу час встановлення залежностей зайняло більше часу ніж минулого разу, що може бути пов’язано, наприклад, з якістю інтернет з’єднання або ще якимись чинниками).

```
maksym@Ubuntu:~/Python$ sudo docker build -t lab3-python:v04 .
[+] Building 48.6s (10/10) FINISHED                               docker:default
 => [internal] load build definition from Dockerfile                        0.0s
 => => transferring dockerfile: 253B                                        0.0s
 => [internal] load metadata for docker.io/library/python:3.12.3-alpine     3.4s
 => [internal] load .dockerignore                                           0.1s
 => => transferring context: 2B                                             0.0s
 => [1/5] FROM docker.io/library/python:3.12.3-alpine@sha256:ef097620baf1   8.4s
 => => resolve docker.io/library/python:3.12.3-alpine@sha256:ef097620baf1   0.0s
 => => sha256:f44387b482817f41bdac1892c45711adaedb3a7dd38 6.02kB / 6.02kB   0.0s
 => => sha256:4abcf20661432fb2d719aaf90656f55c287f8ca915d 3.41MB / 3.41MB   1.4s
 => => sha256:c3cdf40b8bda8e4ca4be0f5fa7f1d128907271e 619.60kB / 619.60kB   0.5s
 => => sha256:3a6cecfe70039fd21206783553d33ea4753700f03 13.96MB / 13.96MB   4.3s
 => => sha256:ef097620baf1272e38264207003b0982285da3236a2 1.65kB / 1.65kB   0.0s
 => => sha256:c583b8590a197db1f6efece2dd244b0259cb6f82c4a 1.37kB / 1.37kB   0.0s
 => => sha256:60d2faee92e78fe7518f0ff1645cd7320bf6b140ff885fd 239B / 239B   1.3s
 => => extracting sha256:4abcf20661432fb2d719aaf90656f55c287f8ca915dc1c92   0.4s
 => => sha256:b62713ed4827911d38bb5a9ac322efa0408b4bb1358 2.70MB / 2.70MB   2.6s
 => => extracting sha256:c3cdf40b8bda8e4ca4be0f5fa7f1d128907271efcbc72cbf   1.3s
 => => extracting sha256:3a6cecfe70039fd21206783553d33ea4753700f031a24904   2.4s
 => => extracting sha256:60d2faee92e78fe7518f0ff1645cd7320bf6b140ff885fde   0.0s
 => => extracting sha256:b62713ed4827911d38bb5a9ac322efa0408b4bb135863b4b   1.2s
 => [internal] load build context                                           0.0s
 => => transferring context: 853B                                           0.0s
```

```
 => [2/5] WORKDIR /app                                                    0.3s
 => [3/5] COPY requirements/requirements.txt /app                         0.1s
 => [4/5] RUN pip install -r requirements.txt                            35.2s
 => [5/5] COPY . /app                                                     0.1s
 => exporting to image                                                    0.8s
 => => exporting layers                                                   0.7s
 => => writing image sha256:1b9653507278a858c5b06c7184d6f395b9952404a48a3  0.0s
 => => naming to docker.io/library/lab3-python:v04                        0.0s
```

Також значно змінився розмір образу, а саме став 143 мегабайти:

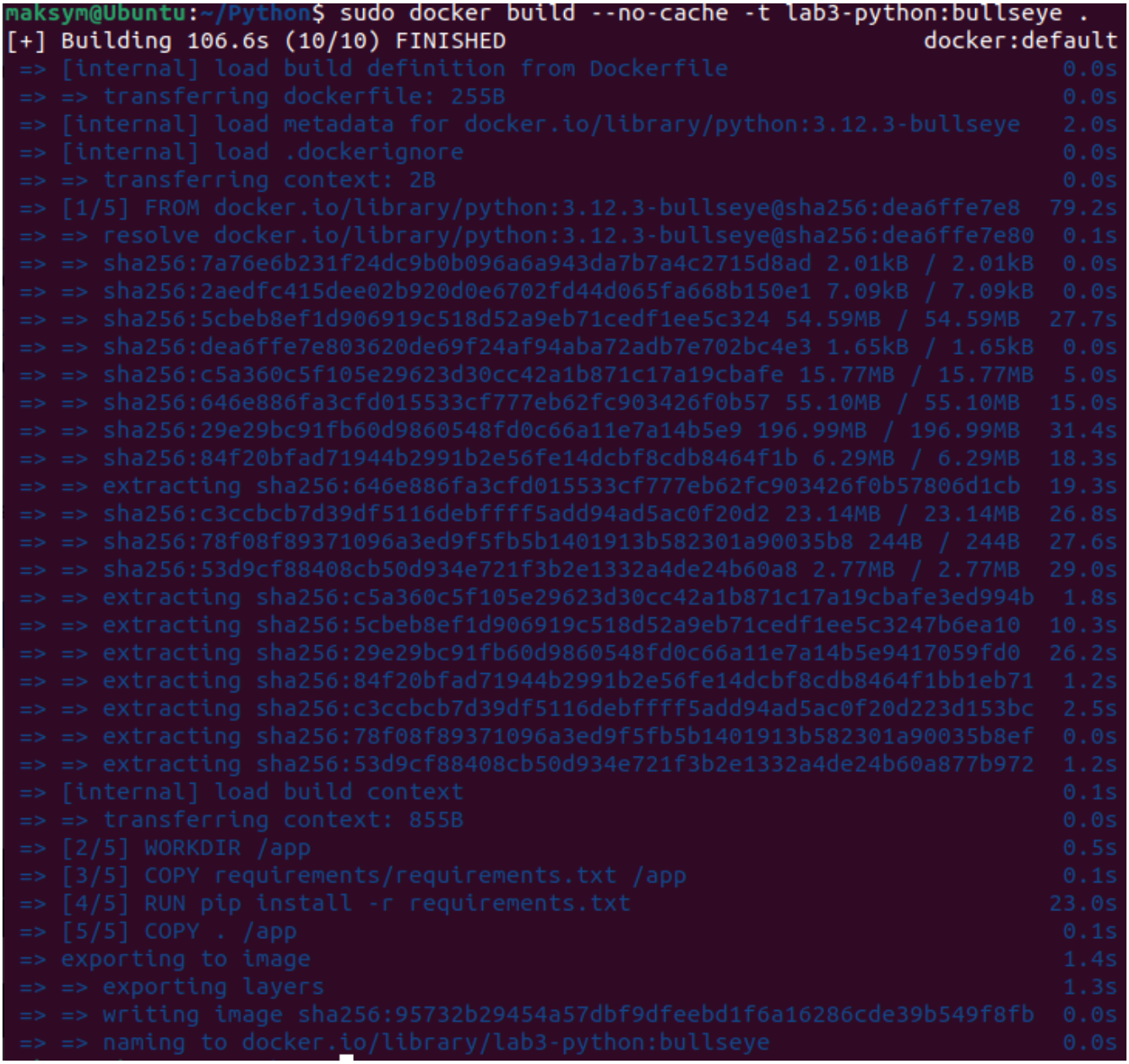
```
maksym@Ubuntu:~/Python$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED              SIZE
lab3-python   v04       1b9653507278   About a minute ago   143MB
lab3-python   v03       742b2fa37cb5   3 hours ago          1.01GB
lab3-python   v02       17377e13d334   4 hours ago          1.01GB
lab3-python   v01       4a6bad1bb34e   4 hours ago          1.01GB
```

5) Додав у файли backend.in та requirements.txt залежність numpy. Далі в spaceship/routers/apy.py додав ендпоінт /api/multiply-matrices з її використанням. Після чого зібрав образ на базовому образі з alpine, а потім – з bullseye (debian) (Перед цим видаливши всі попередні образи).

Образ на alpine зібрався за 38.5 секунд:

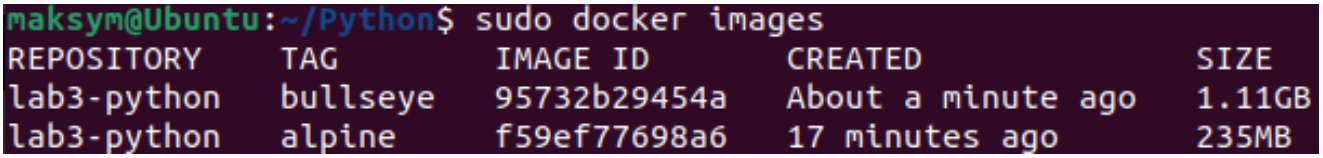
```
maksym@Ubuntu:~/Python$ sudo docker build -t lab3-python:alpine .
[+] Building 38.5s (10/10) FINISHED                              docker:default
 => [internal] load build definition from Dockerfile                      0.0s
 => => transferring dockerfile: 253B                                      0.0s
 => [internal] load metadata for docker.io/library/python:3.12.3-alpine   2.0s
 => [internal] load .dockerignore                                         0.0s
 => => transferring context: 2B                                           0.0s
 => [1/5] FROM docker.io/library/python:3.12.3-alpine@sha256:ef097620baf12  8.1s
 => => resolve docker.io/library/python:3.12.3-alpine@sha256:ef097620baf12  0.0s
 => => sha256:c3cdf40b8bda8e4ca4be0f5fa7f1d128907271ef 619.60kB / 619.60kB  0.3s
 => => sha256:3a6cecfe70039fd21206783553d33ea4753700f031 13.96MB / 13.96MB  2.3s
 => => sha256:ef097620baf1272e38264207003b0982285da3236a20 1.65kB / 1.65kB  0.0s
 => => sha256:c583b8590a197db1f6efece2dd244b0259cb6f82c4ac 1.37kB / 1.37kB  0.0s
 => => sha256:f44387b482817f41bdac1892c45711adaedb3a7dd381 6.02kB / 6.02kB  0.0s
 => => sha256:4abcf20661432fb2d719aaf90656f55c287f8ca915dc 3.41MB / 3.41MB  1.7s
 => => sha256:60d2faee92e78fe7518f0ff1645cd7320bf6b140ff885fde 239B / 239B  1.0s
 => => sha256:b62713ed4827911d38bb5a9ac322efa0408b4bb13586 2.70MB / 2.70MB  2.3s
 => => extracting sha256:4abcf20661432fb2d719aaf90656f55c287f8ca915dc1c92e  0.7s
 => => extracting sha256:c3cdf40b8bda8e4ca4be0f5fa7f1d128907271efcbc72cbfc  1.1s
 => => extracting sha256:3a6cecfe70039fd21206783553d33ea4753700f031a249042  2.2s
 => => extracting sha256:60d2faee92e78fe7518f0ff1645cd7320bf6b140ff885fdec  0.0s
 => => extracting sha256:b62713ed4827911d38bb5a9ac322efa0408b4bb135863b4b1  1.2s
 => [internal] load build context                                         0.1s
 => => transferring context: 11.08kB                                      0.0s
 => [2/5] WORKDIR /app                                                    0.3s
 => [3/5] COPY requirements/requirements.txt /app                         0.1s
 => [4/5] RUN pip install -r requirements.txt                            26.5s
 => [5/5] COPY . /app                                                     0.1s
 => exporting to image                                                    1.3s
 => => exporting layers                                                   1.3s
 => => writing image sha256:f59ef77698a600de48d9b1768fc99dfd7e64c5781ed532  0.0s
 => => naming to docker.io/library/lab3-python:alpine                     0.0s
```

Образ на bullseye (debian) зібрався за 106.6 секунд:



**тут використав опцію вимкнення кешування, щоб виміряти час встановлення залежностей*

Щодо розміру образів, то образ на alpine зайняв 235 мегабайт, в той час як образ на debian – 1.11 гігабайт:



2. Golang

1) Створив Dockerfile для проєкту. Зібрав образ. Час збірки зайняв 25.5 секунд (разом з встановленням базового образу – 11.4 секунди):

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Golang> docker build -t lab3-golang:v01 .
[+] Building 25.5s (11/11) FINISHED
```

Розмір образу становив 320 мегабайт:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Golang> docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
lab3-golang   v01       bb4f6f0df4d2   9 minutes ago    320MB
```

```
    Directory: C:\Users\Maksym\Desktop\lab3-mtrpz\Golang


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        13.05.2024     23:25                cmd
d-----        13.05.2024     23:25                lib
d-----        13.05.2024     23:25                templates
-a----        13.05.2024     23:23           2415 .gitignore
-a----        14.05.2024     13:04            174 Dockerfile
-a----        13.05.2024     23:23            181 go.mod
-a----        13.05.2024     23:23          75705 go.sum
-a----        13.05.2024     23:23            119 main.go
-a----        13.05.2024     23:23           1073 README.rst
```

Також переглянувши вміст директорії можна побачити, що деякі файли непотрібні для копіювання в контейнер, а саме .gitignore, Dockerfile та README.rst. Тому їх можна занести до .dockerignore.

Після [занесення](#) вищезгаданих файлів до .dockerignore, знову зібрав образ. Оскільки базовий образ вже був завантажений, а також деякі етапи закешувались, то збірка зайняла значно менше часу ніж першого разу, а саме <u>8.8 секунди</u>:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Golang> docker build -t lab3-golang:v02 .
[+] Building 0.0s (0/0)  docker:default
[+] Building 8.8s (11/11) FINISHED                                                                   docker:default
 => [internal] load build definition from Dockerfile                                                          0.0s
 => => transferring dockerfile: 213B                                                                          0.0s
 => [internal] load metadata for docker.io/library/golang:1.22.3-alpine                                       1.3s
 => [internal] load .dockerignore                                                                             0.0s
 => => transferring context: 76B                                                                              0.0s
 => [1/6] FROM docker.io/library/golang:1.22.3-alpine@sha256:2a882244fb51835ebbd8313bffee83775b0c076aaf56b497b43d8a4c72db65e1  0.0s
 => [internal] load build context                                                                             0.0s
 => => transferring context: 400B                                                                             0.0s
 => CACHED [2/6] WORKDIR /app                                                                                 0.0s
 => CACHED [3/6] COPY go.mod go.sum ./                                                                        0.0s
 => CACHED [4/6] RUN go mod download                                                                          0.0s
 => [5/6] COPY . .                                                                                            0.0s
 => [6/6] RUN go build -o build/fizzbuzz                                                                      7.0s
```

Щодо розміру образу, то він залишився практично незмінним оскільки файли, які були занесені до .dockerignore, не займали багато місця. Але всеодно така дія з .dockerignore є корисною, бо щонайменше контейнер не буде містити зайвих файлів.

2) Створив [Dockerfile](#) з багатоетапною збіркою, зібрав образ. Час збирання образу не мав суттєвої різниці з минулою збіркою, а от розмір образу став набагато меншим, а саме <u>10.7 мегабайт</u>:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Golang> docker images
REPOSITORY    TAG     IMAGE ID       CREATED         SIZE
lab3-golang   v03     5b37e580ef34   2 hours ago     10.7MB
lab3-golang   v02     4ed95834cc35   11 hours ago    320MB
lab3-golang   v01     bb4f6f0df4d2   11 hours ago    320MB
```

Такі зміни в розмірі пов’язані з тим, що цього разу збірка образу була поділена на два етапи, де в першому виконувались всі дії для збирання двійкового файлу, тобто встановлення операційної системи, залежностей і т.п., а на другому етапі цей двійковий файл та каталог templates, який містить html-файл, було скопійовано в порожній образ (спочатку я скопіював тільки двійковий файл, але образ не запускався, тому було скопійовано також html-файл), тобто без всіх завантажень з попереднього етапу.

Але оскільки тепер контейнер містить тільки файл fizzbuzz та каталог з html-файлом, то це говорить про те, що тепер стає неможливою робота з самим вмістом контейнера, тобто, наприклад, навіть не вийде в контейнері виконати команду ls, оскільки там немає ніякої оболонки для виконання цієї команди. Тобто таким способом зручно користуватись, якщо не потрібно додатково працювати всередині контейнеру, а достатньо просто запустити образ.

3) Створив Dockerfile з багатоетапною збіркою, але цього разу на другому етапі використав базовий образ distroless замість scratch.
Збірка образу з нуля зайняла 25.1 секунду:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Golang> docker build -f .\distroless.Dockerfile -t lab3-golang:v04 .
[+] Building 0.0s (0/0)  docker:default
[+] Building 25.1s (15/15) FINISHED
```

Розмір образу став трохи більшим в порівнянні з попереднім образом, а саме 13.3 мегабайт, але все ще такий розмір є набагато меншим ніж у образів з одноетапною збіркою.

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Golang> docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
lab3-golang   v04       eac28ac8157a   4 minutes ago   13.3MB
```

Як і з образом scratch тут також не можна виконувати дії всередині контейнера, оскільки так само немає пакетних менеджерів, оболонок або ще якихось програм. Але наскільки я зрозумів, то в порівнянні з повністю пустим scratch, тут все-таки є деякі базові компоненти, які можуть знадобитись для роботи наприклад з користувачами, сертифікатами або ще можна додати якісь додаткові налаштування, а також є інші версії distroless-образів, які можна підбирати в залежності від потреб. Тобто, знову ж таки, вибір способу збірки залежить від потрібного для роботи функціоналу.

3. Javascript

Створив простий застосунок, який виводить в браузері дату та час, перед цим привітавшись по імені, яке було передано в аргумент при запуску програми.
Спочатку написав Dockerfile на базі образу **node:22-bullseye**.
Збірка такого образу зайняла 50.3 секунди:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Javascript> docker build -t lab3-javascript:v01 .
[+] Building 48.1s (10/10) FINISHED
```

А розмір досягнув 1.02 гігабайти:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Javascript> docker images
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
lab3-javascript   v01       479619f81739   2 minutes ago   1.02GB
```

Робота застосунку при запуску контейнера була коректною:

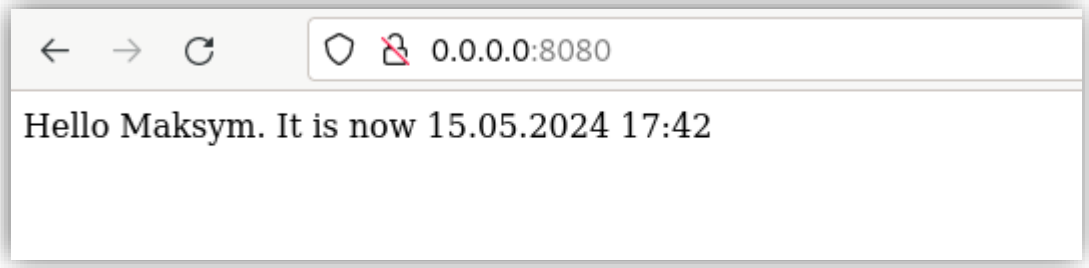
```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Javascript> docker run -p8080:8080 --rm lab3-javascript:v01

> javascript@1.0.0 start
> node app.js

Please provide a name as an argument.
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Javascript> docker run -p8080:8080 --rm lab3-javascript:v01 Maksym

> javascript@1.0.0 start
> node app.js Maksym

Server running at http://0.0.0.0:8080/
```



Потім змінив в Dockerfile базовий образ на **node:22-alpine**, бо як вже виявилось з попередніх експерментів він значно менше за розміром за bullseye.
З цим образом час збірки зменшився до 10 секунд:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Javascript> docker build -t lab3-javascript:v02 .
[+] Building 10.0s (10/10) FINISHED
```

А розмір став 156 мегабайт:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Javascript> docker images
REPOSITORY        TAG       IMAGE ID       CREATED              SIZE
lab3-javascript   v02       a78058966202   About a minute ago   156MB
```

Робота застосунку залишилась незмінною.
Потім вирішив спробувати переробити Dockerfile для багатоетапної збірки. На другому етапі використав образ **gcr.io/distroless/nodejs22-debian12**, оскільки він має підтримку node.js, але проблема полягала в тому, що наскільки я розумію в цьому образі за замовчуванням в ENTRYPOINT використовується “node” або щось подібне, тобто мені потрібно писати в CMD назву файлу, який треба запустити. Але проблема в тому, що оскільки моя програма потребує аргументу разом з командою для запуску контейнера, то це говорить про те, що мені для коректного запуску програми треба в аргументи передавати app.js та ім’я (тобто сам аргумент, який потрібен), оскільки аргументи передані команді в консолі замінюють вміст CMD.

Та ще й плюс до цього такий спосіб збірки (з цим образом) займав 193 мегабайти, що більше ніж з минулою одноетапною збіркою. Щодо використання інших образів, де можна використати команду “node” напряму, тобто встановити її туди, наприклад той же alpine, то я вирішив, що краще тоді більше оптимізувати саме одноетапну збірку, бо з такої багатоетапної збірки не буде особливого сенсу. Можливо і можна якось було вигадати використання багатоетапної збірки в цьому випадку, яке було б краще одноетапної, але в мене не вийшло.

Тому я повернув Dockerfile до одноетапної збірки та вніс ще деякі зміни. По-перше під час минулих збірок я помітив, що я не можу закрити сервер в консолі за допомогою Ctrl + C. Пошукавши про це інформацію в інтернеті, я знайшов, що це доволі часто пов’язано з особливістю роботи Docker. Я не сильно в це заглиблювався, але проблема в тому, що в деяких випадках (доволі часто в Node js), система може не реєструвати сигнали, наприклад, такі як SIGNTERM. Я знайшов, що для коректної роботи з цими сигналами, часто використовують dumb-init, тому я використав її у своєму Dockerfile і в мене запрацювала комбінація Ctrl+C. Окрім цього я також переробив встановлення node js у контейнер, а саме цього разу як базовий образ я використав просто alpine (без node js), а вже далі за допомогою RUN встановив node js, оскільки в процесі того як я шукав вирішення минулих проблем, я помітив, що такий спосіб допомагає зменшити розмір зібраного образу.

Після внесених змін в Dockerfile, образ зібрався за 10.6 секунд:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Javascript> docker build -t lab3-javascript:v03 .
[+] Building 10.6s (12/12) FINISHED
```

А його розмір склав 81 мегабайт:

```
PS C:\Users\Maksym\Desktop\lab3-mtrpz\Javascript> docker images
REPOSITORY        TAG       IMAGE ID       CREATED          SIZE
lab3-javascript   v03       4bcc61c28876   45 seconds ago   81MB
```

Це майже в два рази менше ніж просто з **node:22-alpine**.

Щодо .dockerignore файлу, то я вирішив його не використовувати, оскільки в мене код застосунку знаходиться тільки в одному .js файлі, тому я тільки його копіюю, а список залежностей також окремо копіюється. Тому в контейнері знаходяться тільки потрібні файли.

Висновок

Після проведених експериментів, я можу сказати, що вибір способу збірки Docker-образу залежить від самого застосунку та того як образ буде використовуватись.

Велике значення для збірки має базовий образ, бо як виявилось це може значно вплинути як і на тривалість збірки, так і на розмір кінцевого образу. Наприклад, образи, які базуються на bullseye будуть займати значно більше місця та довше збиратись ніж образи на alpine.

При написанні докерфайлу треба уважно слідкувати за порядком виконання команд, оскільки від цього залежить чи будуть повторюватись дії, які вже і так були виконані раніше, чи вони зможуть закешуватись і тим самим значно прискорити збірку. Потрібно спочатку додавати в образ те, що змінюється найрідше і потім вже те, що частіше.

Для великих і тяжких застосунків може гарно підійти багатоетапна збірка, особливо якщо на останніх етапах використовувати дуже малі образи такі як scratch (пустий образ) або distroless (з деякими базовими компонентами), то можна значно зменшити розмір кінцевого образу. Але такі невеликі образи краще використовувати, коли його можна запустити там без купи інших встановлень, а також коли його потрібно лише запустити, тобто не потрібно додатково працювати з вмістом контейнера. Якщо потрібно більше функціоналу, то можна використовувати інші образи, наприклад alpine, який вже має більше можливостей, а його розмір також достатньо малий. Якщо ж застосунок є невеликим, то можливо немає сенсу використовувати багатоетапну збірку, або ж навіть краще використати одноетапну.

Також може бути дуже корисним використання файлу .dockerignore, в який можна внести файли, які будуть ігноруватись докером, тобто які не потрібні для копіювання в контейнер. Цим самим залишивши в контейнері тільки потрібне, а також, хоч можливо і не суттєво, але зменшити розмір образу.