**Лабораторна робота №3**

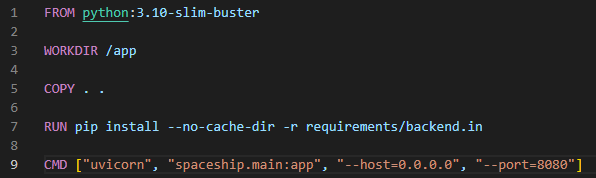
**Контейнеризація**

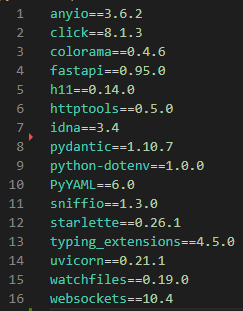
**Семиволос Євгеній ІМ-11**

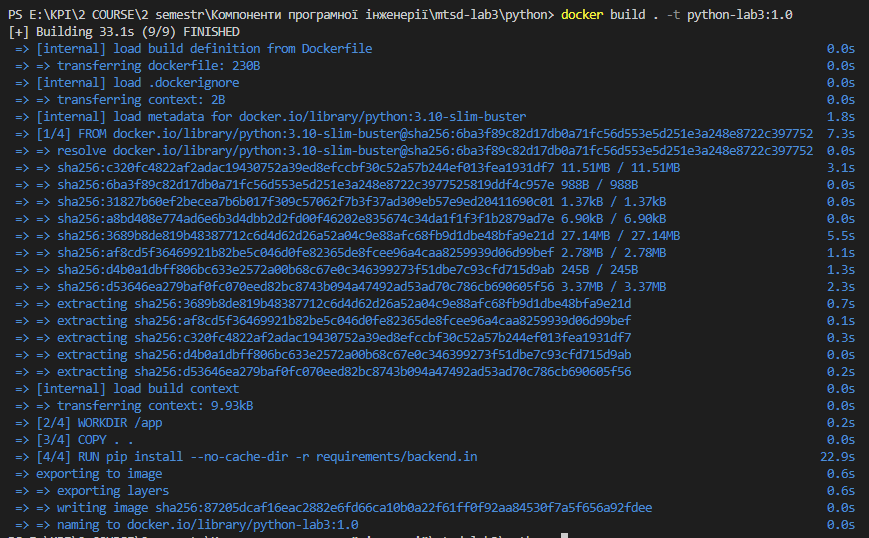
**Python**

1. Створюємо опис контейнера для застосунку. В образі буде код застосунку та залежності (використано *pip freeze*).

*Dockerfile:*

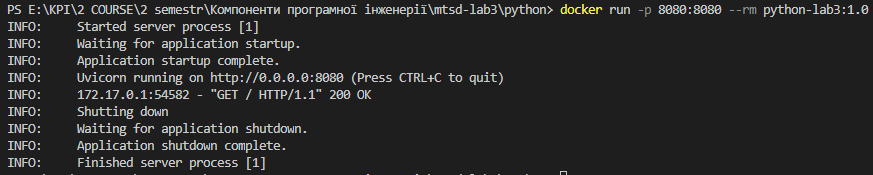


*Dependencies:*  


*Build:* **docker build . -t python-lab3:1.0**

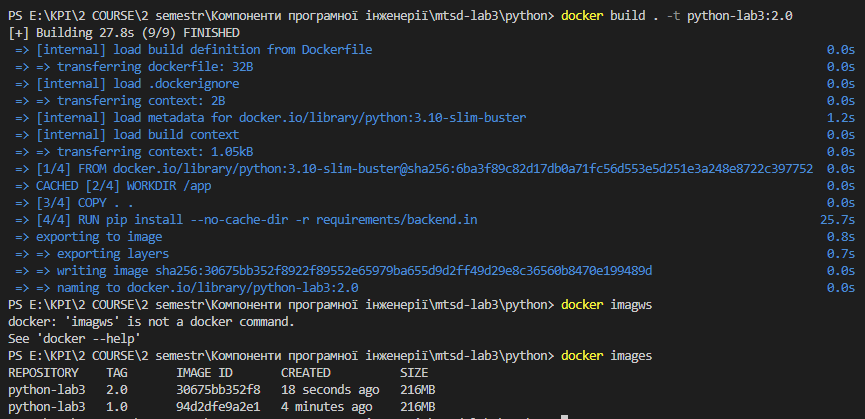
*Build time:* **33.1s**

*Size:* **216 MB**

Run: **docker run -p 8080:8080 --rm python-lab3:1.0**

[**link**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/1b7125338ac488688741c60cb2f9175d42e1f111) **on commit**

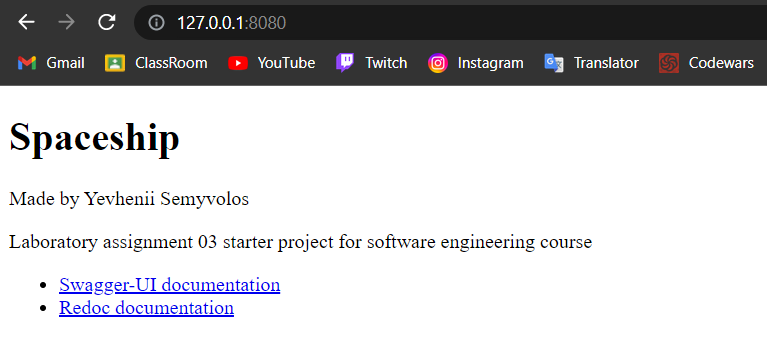
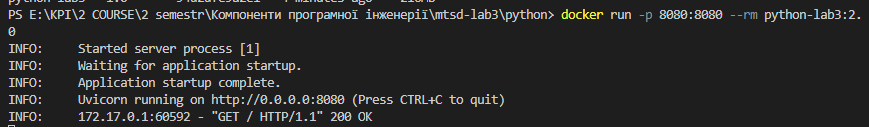
Змінюємо вміст build/index.html

*Build:* **docker build . -t python-lab3:2.0**

*Build time:* **27.8s** (трохи швидше, бо деякі налаштування були взяті з кеша)

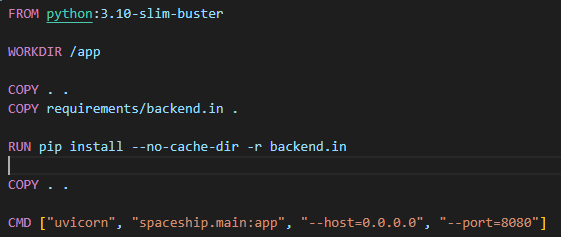
*Size:* **216 MB**

**docker run -p 8080:8080 --rm python-lab3:2.0**

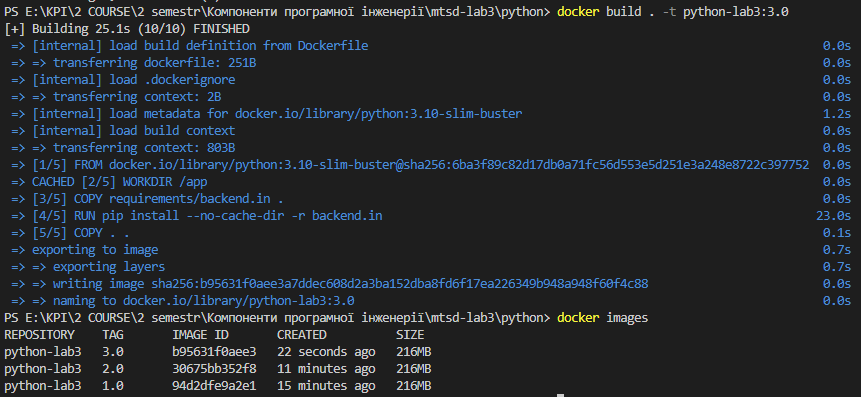


2. Оптимізація Dockerfile

Поки Docker працює неоптимізовано. Щоб оптимізувати, треба ефективно використовувати шари (layers) для збірки образу. Для цього спершу додаємо у образ те, що змінюватиметься найрідше (системні залежності, залежності проекту) а ті файли, які змінюватимуться частіше (наш код) — додаємо останніми.

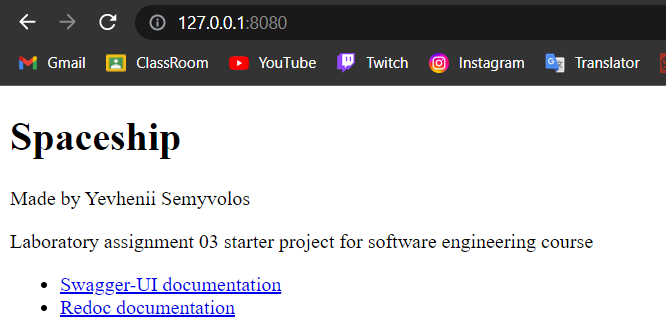
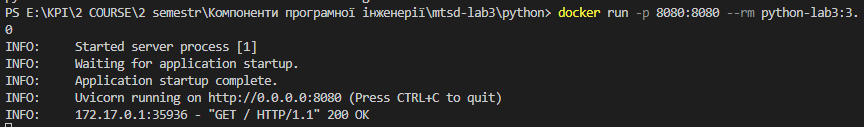
Dockerfile:  


Build: **docker build . -t python-lab3:3.0**



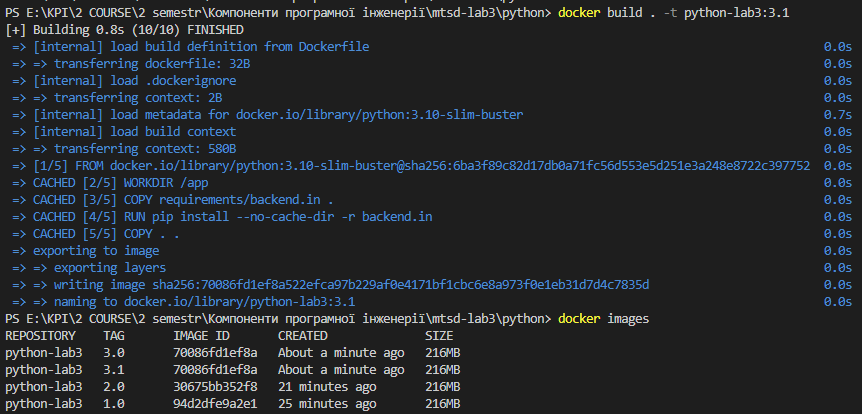
*Build time:* **25.1s**

*Size:* **216 MB**

Run: **docker run -p 8080:8080 --rm python-lab3:3.0**

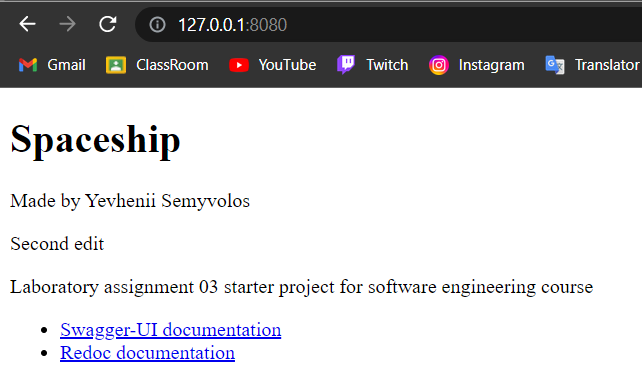
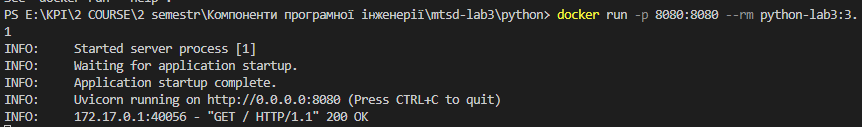
[**link**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/fd882b083011aaa4863191391e7f7f82a4671436) **on commit**

Змінюємо вміст build/index.html

*Build:* **docker build . -t python-lab3:3.1**

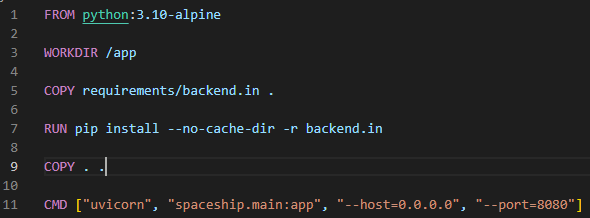
*Build time:* **0.8s** (значно швидше завдяки кешуванню)

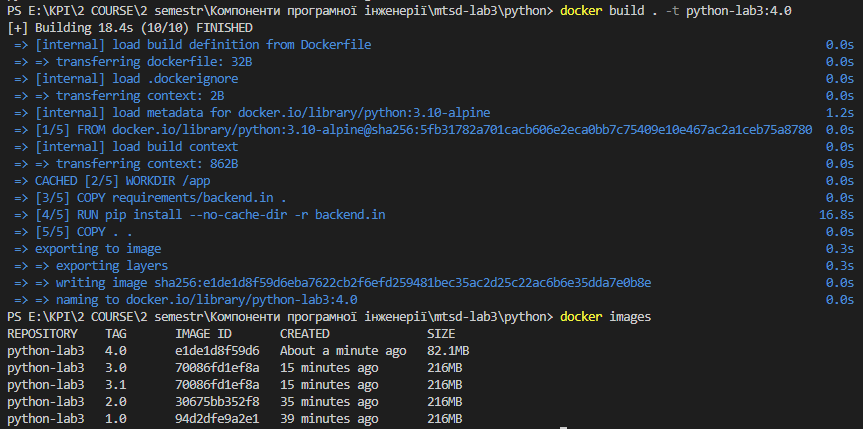
*Size:* **216 MB**

*Run:* **docker run -p 8080:8080 --rm python-lab3:3.1**

1. Змінюємо базовий образ:

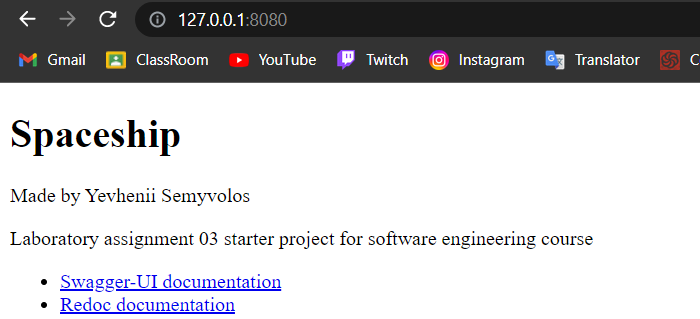
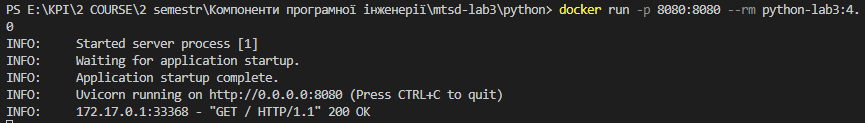
Dockerfile:



*Build:* **docker build . -t python-lab3:4.0**

*Build time:* **18.4s**

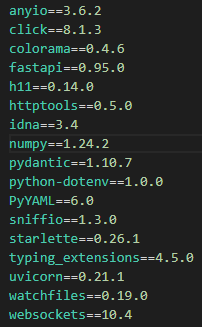
*Size:* **82.1 MB** (alpine менш пам’яті займає, ніж slim-buster)

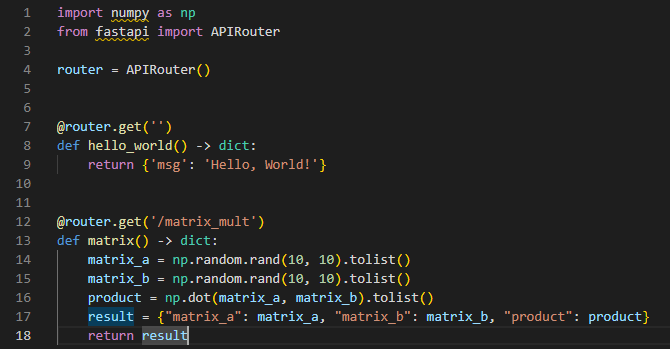
*Run:* **docker run -p 8080:8080 --rm python-lab3:4.0**

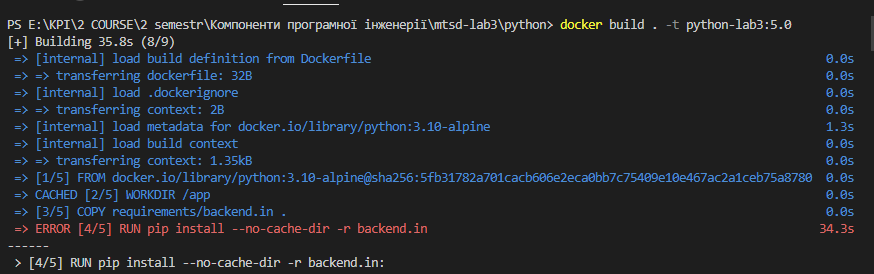
[**link**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/b1251debd520903283c197146702390da76ccf8b) **on commit**

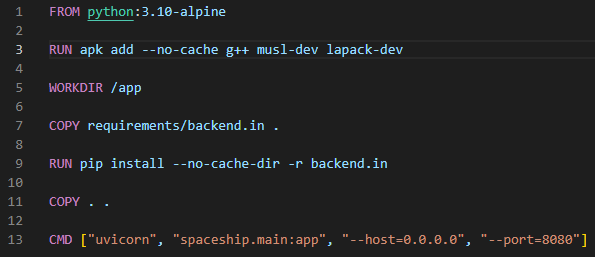
1. Додаємо залежність numpy та ендпоінт, який згенерує 2 випадкові матриці 10x10 та перемножить їх між собою.

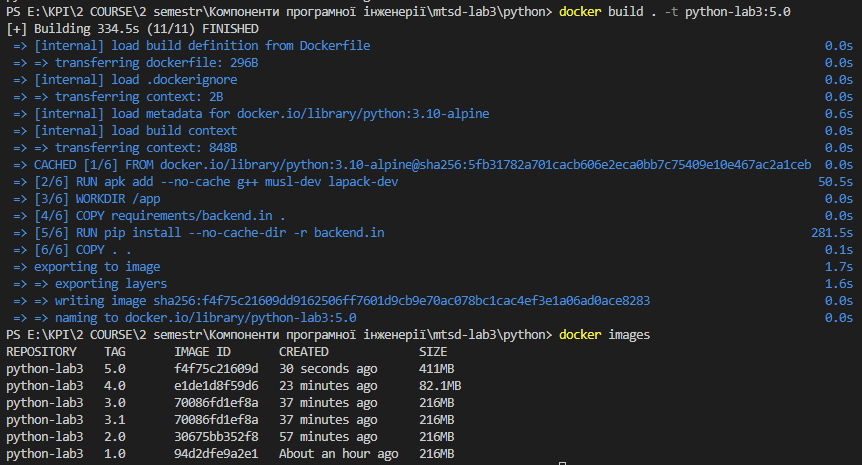
Dependencies:



spaceship/routers/api.py:

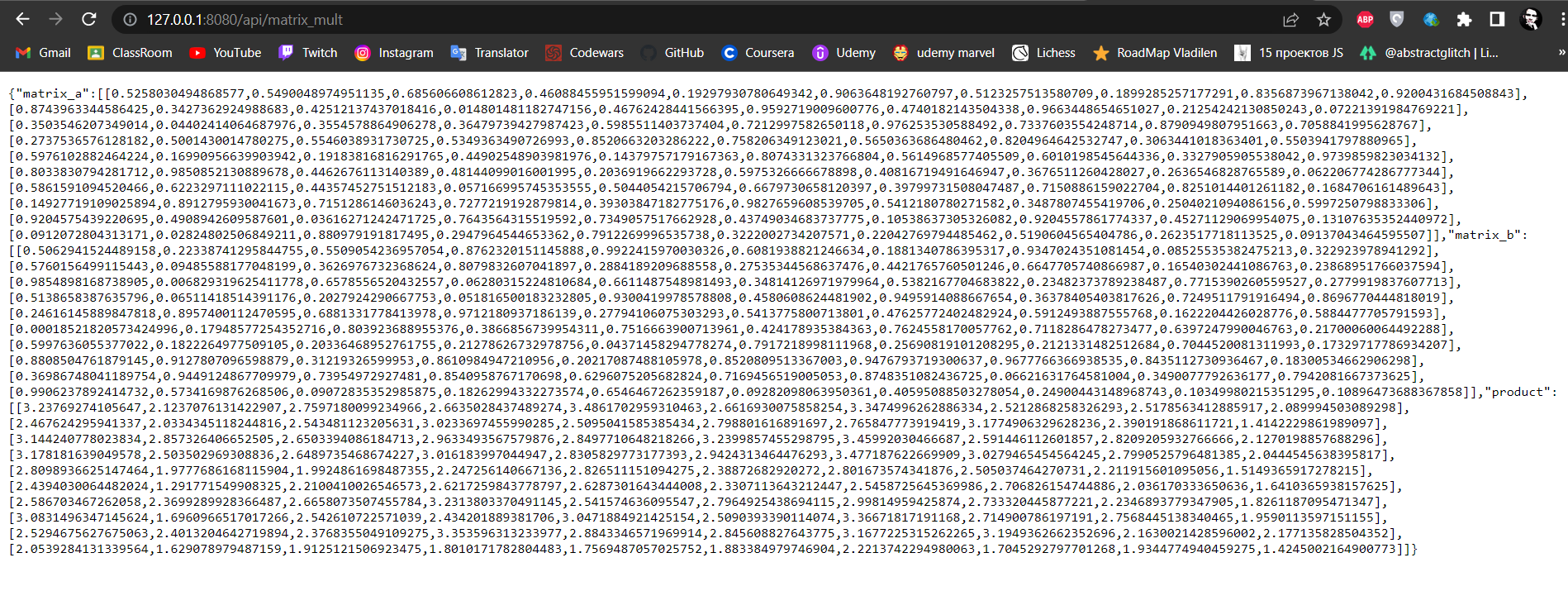
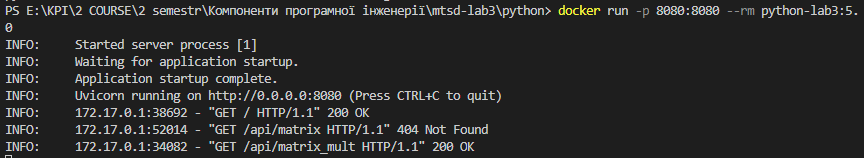
*Build:* **docker build . -t python-lab3:5.0**  


При додаванні numpy при білді видає помилку. Треба підправити Dockerfile:  


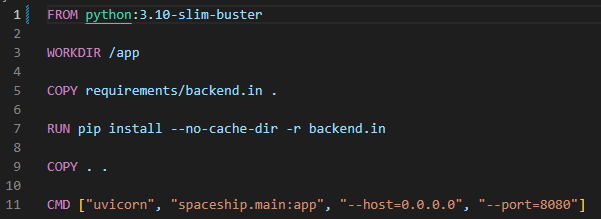
*Build:* **docker build . -t python-lab3:5.0**

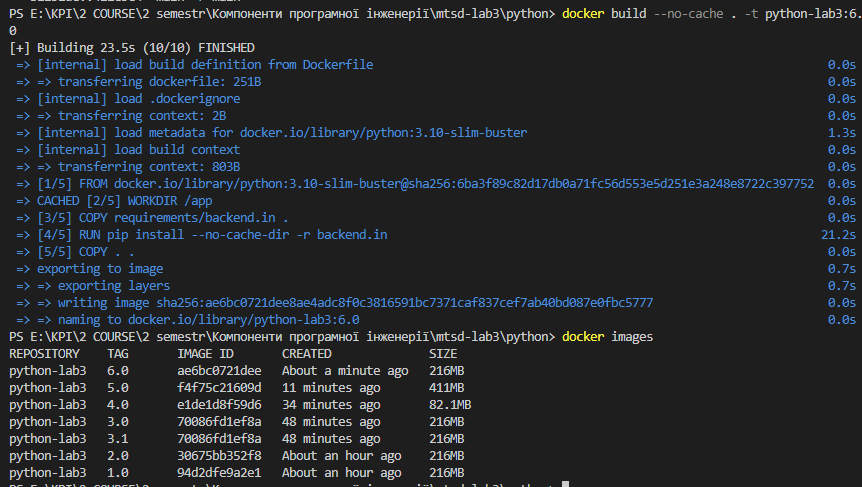
*Build time:* **334.5s** (завантаження нових залежностей)

*Size:* **411 MB**

Run: **docker run -p 8080:8080 --rm py-lab3:5.0**

**link on** [**commit1**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/421e5e4eb047cf82fe9c445bfb61b8ae08067308) **and** [**commit2**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/b5212bd55c39f3b9749719bcfbe491b877d5e5dd)

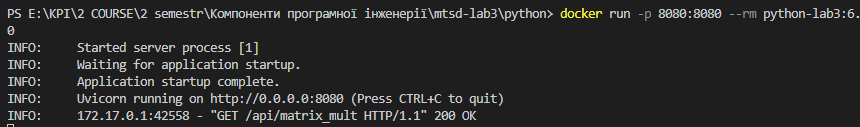
5. Повертаємося до збірки на debian:  


*Build:* **docker build --no-cache . -t python-lab3:6.0**

*Build time:* **23.5s**

*Size:* **216 MB**

*Run:* **docker run -p 8080:8080 --rm python-lab3:6.0**



**link on** [**commit**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/ca1b122ffc1c2f398ac4a5ad5cd192d2a40679b5)

Використання образу на базі alpine потребувало більше часу та зусиль на встановлення додаткових залежностей для підтримки numpy, що зайняло більше часу та збільшило розмір образу.

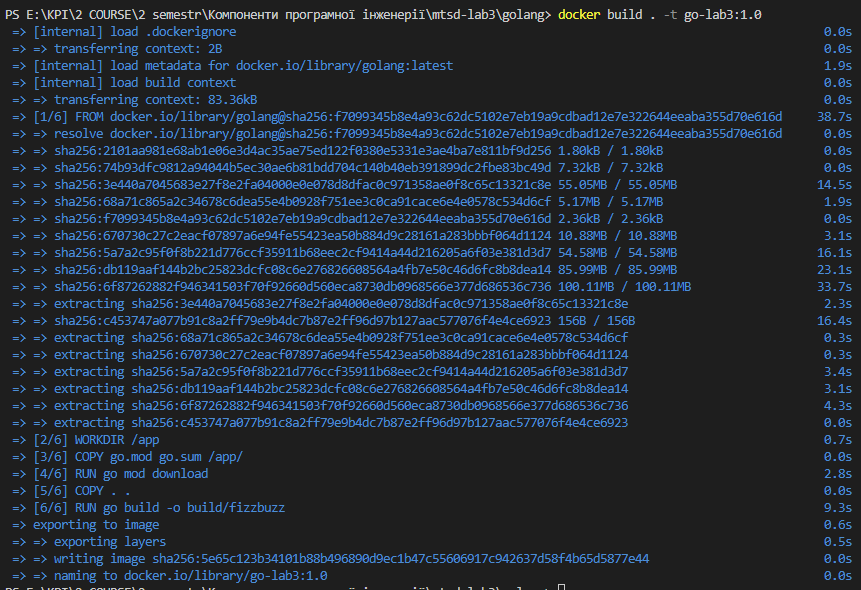
**Golang**

1.

Dockerfile:



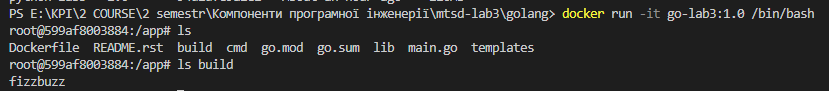
*Build:*  **docker build . -t go-lab3:1.0**





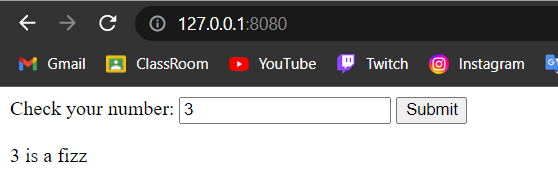
**Build time: 104.9s**

**Size: 861 MB**

Усі присутні файли:

Для запуску необхідний лише бінарний файл /build/fizzbuzz, який був створений командою *go build -o build/fizzbuzz* у Dockerfile

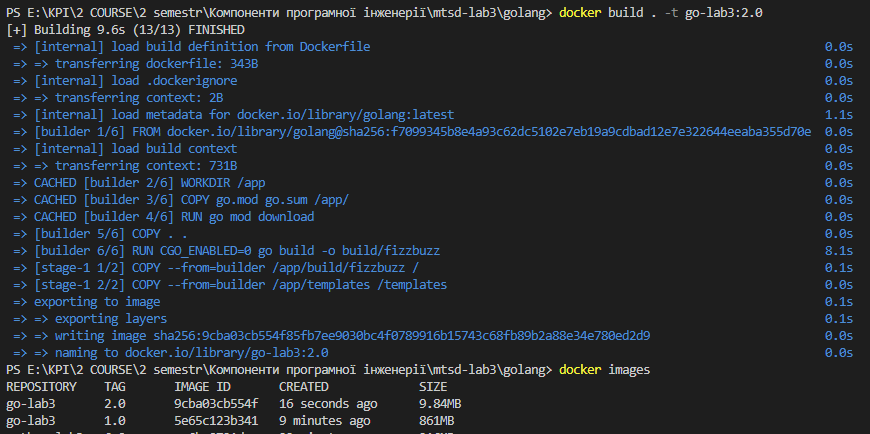
Run: **docker run -p 8080:8080 --rm go-lab3:1.0**

**link on** [**commit**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/8d795aff80fc7dac6fd0c47f0a7dd8ef24bbf07b)

2. Робимо багатоетапну збірку на базі scratch

Dockerfile:  

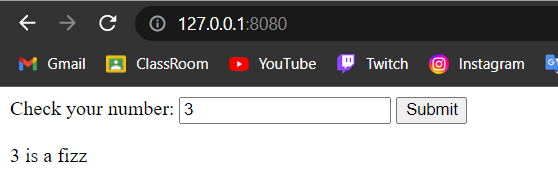

Build: **docker build . -t go-lab3:2.0**

**Build time: 9.6s**

**Size: 9.84 MB** (тут лише потрібні файли)

*Run:* **docker run -p 8080:8080 --rm go-lab3:2.0**



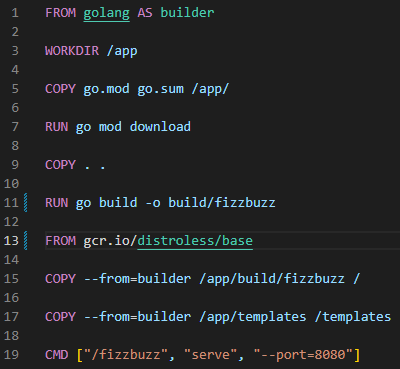


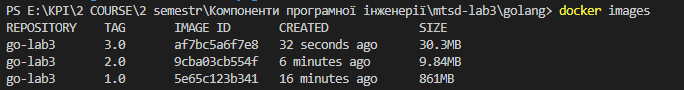
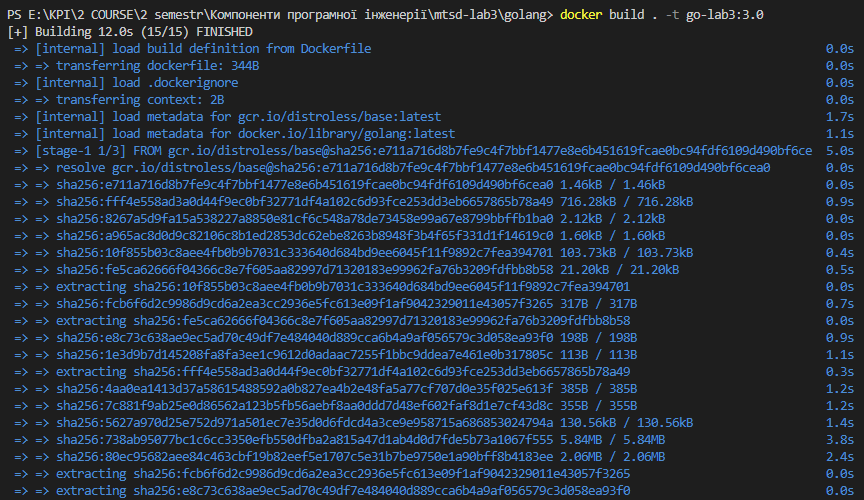
**link on** [**commit**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/728e39dadd1c2f82ef728d08e6e4e8820f1f622f)

Для запуску файлів достатньо. Виконуваний файл скомпільовано статично та всі залежності включені у нього В образ на базі scratch копіюються тільки виконуваний файл та html сторінка.

Багатоетапна збірка образу на базі scratch виявилась корисною, щоб не включати в образ зайві залежності та файли, які не потрібні для запуску. Це значно зменшило розмір образу та швидкість збірки.

3. Використаємо distroless замість scratch:

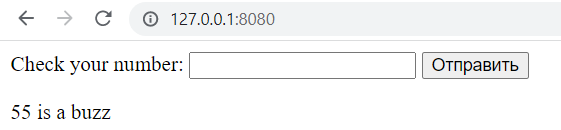
Dockerfile:  


Build: **docker build . -t go-lab3:3.0**

**Build time: 12s** **Size: 30.3 MB**

Run: **docker run -p 8080:8080 --rm go-lab3:3.0**

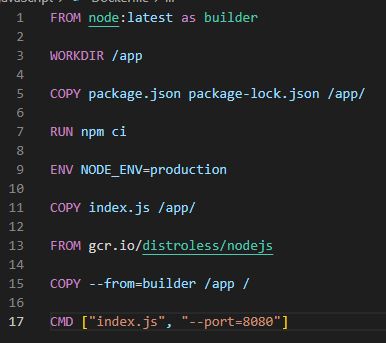
****

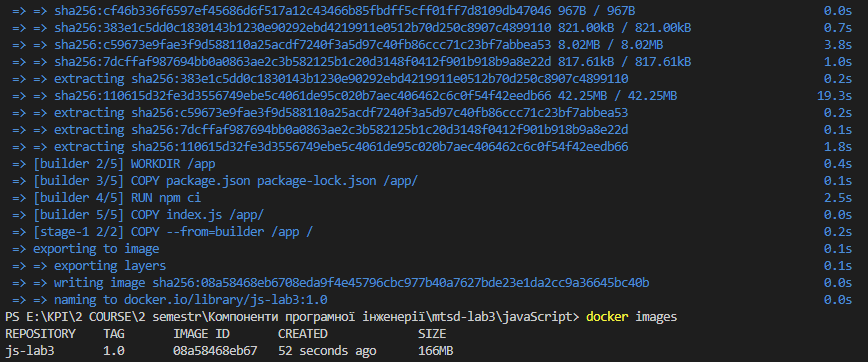
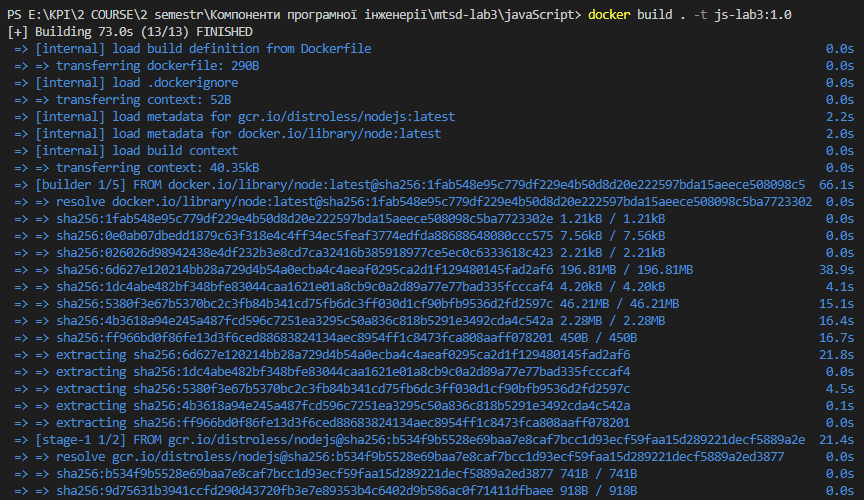
****

**link on** [**commit**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/4d198984330bcdb3f3aacc9855b6e0bf03f886d0)

Образи distroless мають високу безпеку та вже містять тільки необхідні бібліотеки та ніяких інших компонентів системи. Тому розмір такого образу в даному випадку трохи більший за scratch.

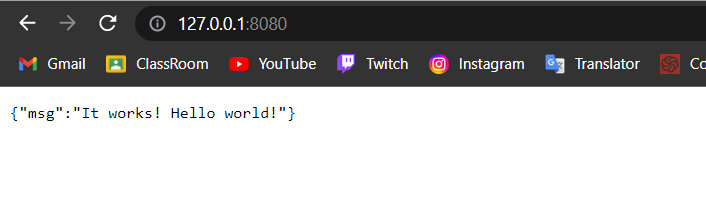
**JavaScript**

1. Dockerfile  
   

*Build*: **docker build . -t js-lab3:1.0**

**Build time: 73s** **Size: 166 MB**

*Run*: **docker run -p 8080:8080 --rm js-lab3:1.0**



**link on** [**commit**](https://github.com/EugeneSemivolos/mtsd-lab3/commit/f05ff3b30f0371195a5c0c35174b00d82f797ec7)

**Висновок**

У даній роботі я досліджував, як працює Docker для різних застосунків на мовах Python, Golang та JavaScript.

Python  
Отримав результат, що у деяких випадках alpine краще по часу та пам’яті, однак коли додавав numpy, виникали труднощі, тож збірка на debian є більш практичною ніж на alpine

Golang  
Багатоетапна збірка значно зменшила розмір образу, бо додавалися тільки необхідні файли. Якщо порівнювати scratch та distroless, то налаштування distroless є простішим, бо тут вже є необхідні бібліотеки. Це трохи збільшує розмір файлу, порівнюючи з scratch, але для маленького проекту це не критично.

JS

Закріпив знання.

Також помітив, що вибір базового образу має велике значення. Треба звертати увагу на розмір та інші особливості (наприклад операційна система).