**Міністерство освіти і науки України  
 Національний технічний університет України  
 «Київський політехнічний інститут імені Ігоря Сікорського»  
 Факультет інформатики та обчислювальної техніки  
 Кафедра обчислювальної техніки**

**Лабораторна робота №3**

з дисципліни  
 «Методології та технології розробки програмного забезпечення»

Виконала: Перевірив:

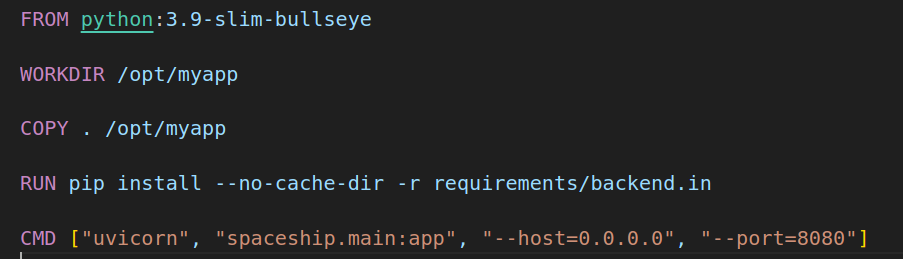
Студентка групи: ІМ-21 Ковальчук О.М.  
Рабійчук Дар’я Олександрівна

номер у списку групи: 18

**Київ 2024**

1. **Python**

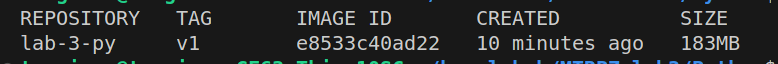
**Збірка №1:**



**Час:** 6.97 с



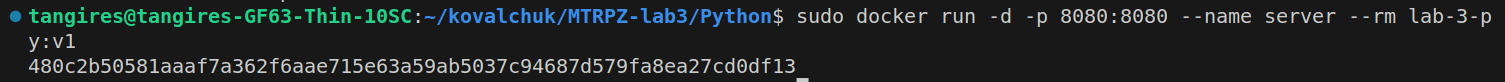
**Розмір:** 183MB



**Запускаємо:**

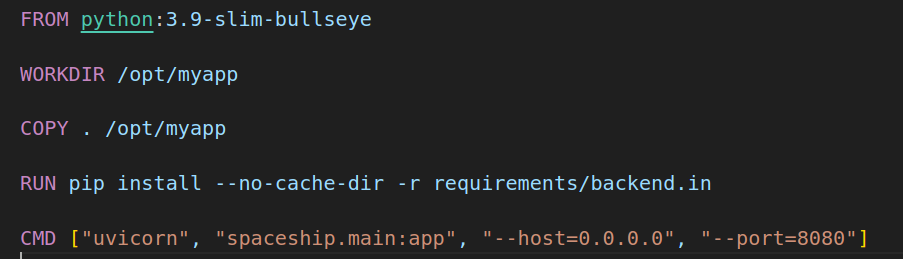
sudo docker run -d -p 8080:8080 --name server --rm lab-3-py:v1

**Отримуємо:**



**Висновок:** збірка неоптимізована.

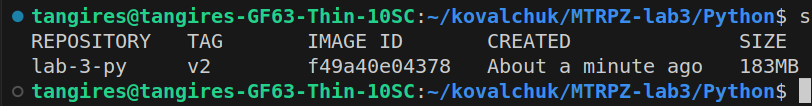
**Збірка №2:**



**Час:** 3.42 с

****

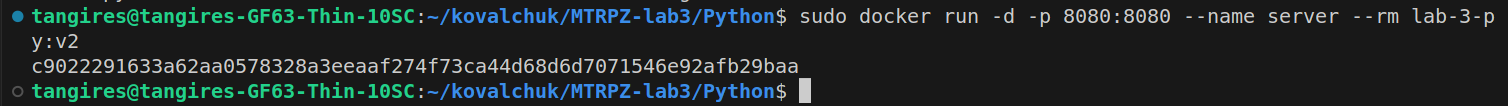
**Розмір:** 183MB

****

**Запускаємо:**

sudo docker run -d -p 8080:8080 --name server --rm lab-3-py:v2

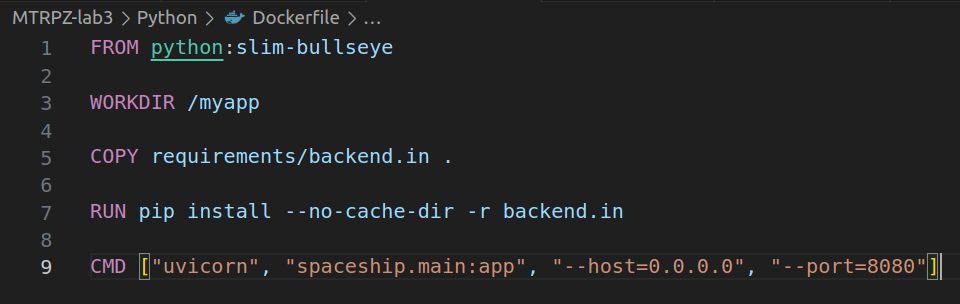
**Отримуємо:**

****

**Висновок:**Ми використали готовий образ python:slim-bullseye з кешу.Через це час збірки скоротився майже у 2 рази. Образ не змінився в розмірах.

.

**Збірка №3**

****

**Розмір:** 183MB

****

**Час:** 7.17 с

****

**Запускаємо:**

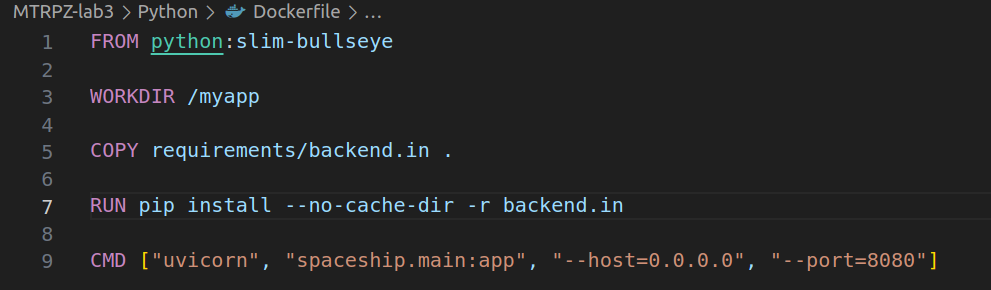
sudo docker run -d -p 8080:8080 --name server --rm lab-3-py:v3

**Отримуємо:**

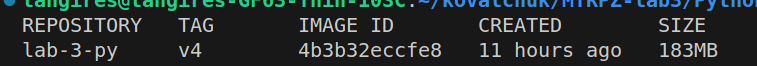
**Висновок:**

Видалення рядка `COPY . .` збільшує час збірки образу майже вдвічі. Це означає, що файли, що знаходяться в поточній директорії, не будуть скопійовані в образ Docker. Це може призвести до втрати певних файлів або залежностей, які можуть бути необхідними для коректної роботи нашого застосунку.

**Збірка №4**

****

**Розмір:** 183MB

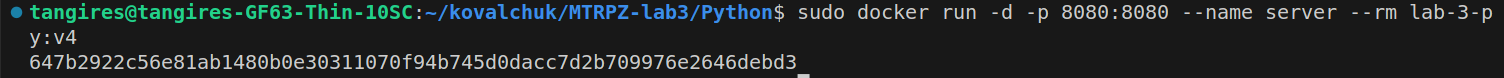
****

**Час:** 1.35 с

****

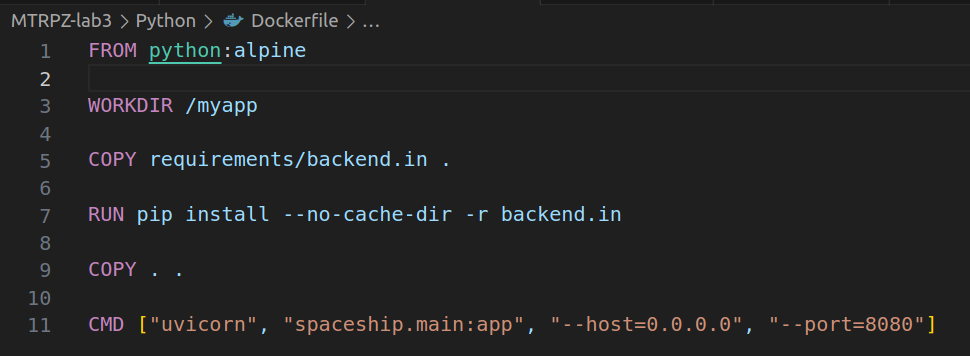
**Запускаємо:**

sudo docker run -d -p 8080:8080 --name server --rm lab-3-py:v4

**Отримуємо:**

**Висновок:**Бачимо, що використання кешування суттєво прискорює процес збірки образу Docker, адже при додаванні тексту до проекту збірка призвела до значного скорочення часу з 7.17 секунд до 1.35 секунд.

**Збірка №5**

****

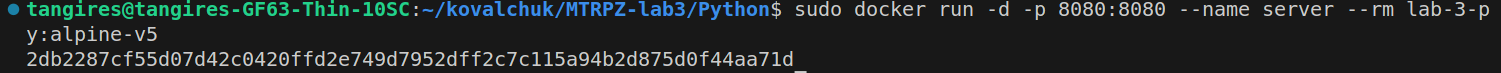
**Розмір:** 112MB ****

**Час:** 21,16 с

**Запускаємо:**

sudo docker run -d -p 8080:8080 --name server --rm lab-3- python:alpine-matrix-v5

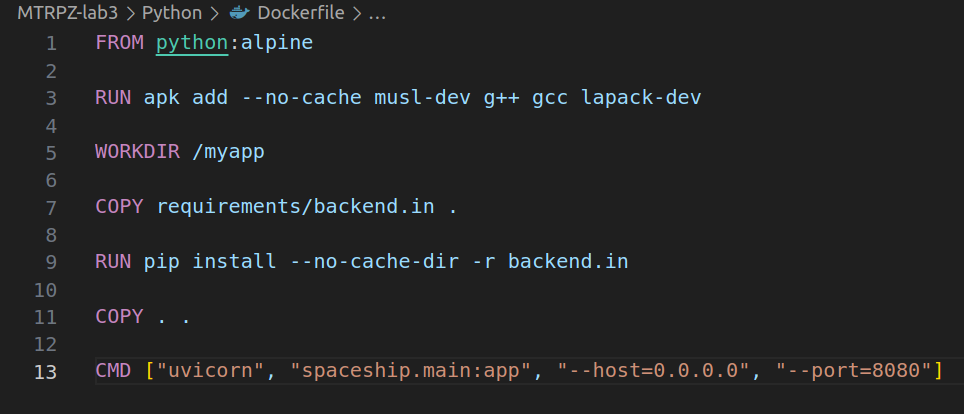
**Отримуємо:**



**Висновок:**

Образ Alpine має розмір 112 мегабайт, що значно менше, ніж образ на основі slim-bullseye, розмір якого складає 182 мегабайти. Використання базового образу Alpine дозволяє створювати легкі та компактні Docker-образи. Час збірки образу за 21,16 секунд свідчить про ефективність Alpine як базового образу. Таким чином, використання базового образу Alpine є ефективним вибором для створення компактних та швидких Docker-образів.

**Збірка №6**

****

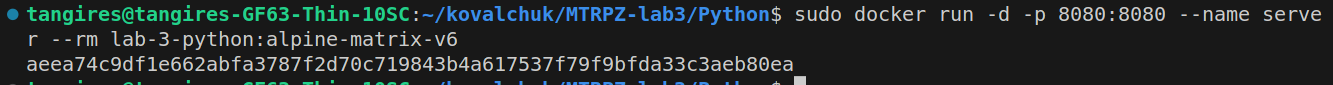
**Розмір:** 442MB

****

**Час:** 269 с

**Запускаємо:**

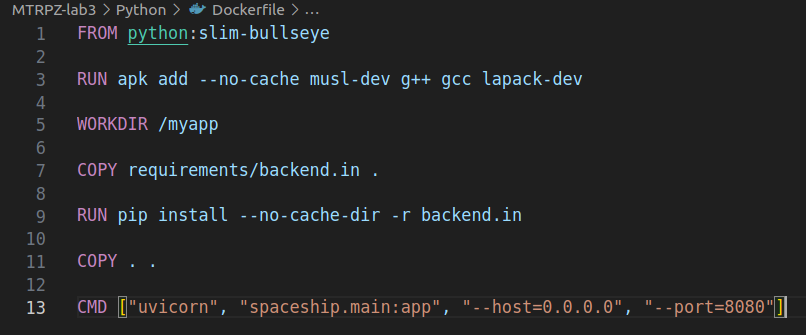
sudo docker run -d -p 8080:8080 --name server --rm lab-3-python:alpine-matrix-v6

**Отримуємо:**

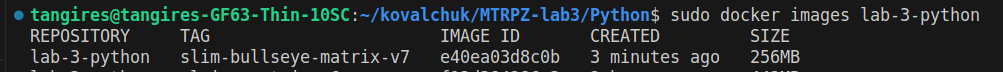
**Висновок:**

Результати експерименту показали, що образ на базі Alpine значно більший і потребує більшого часу для збірки у порівнянні з образом на базі Slim-Bullseye. Встановлення необхідних залежностей вручну на базі Alpine виявилося не дуже зручним. Отже, вибір оптимального образу для контейнеру може допомогти економити час і ресурси.

**Збірка №7**

****

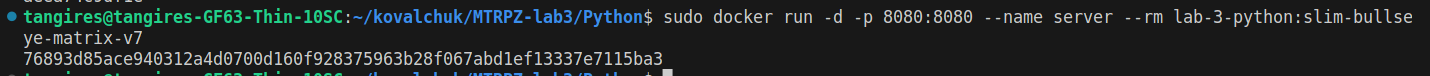
**Розмір:** 256MB



**Час:** 21,6 с

**Запускаємо:**

sudo docker run -d -p 8080:8080 --name server --rm lab-3- python:slim-bullseye-matrix-v7

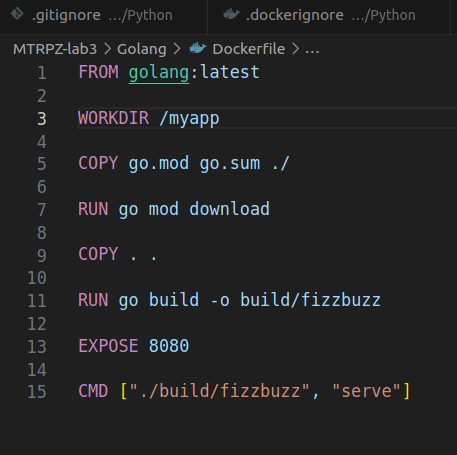
**Отримуємо:**

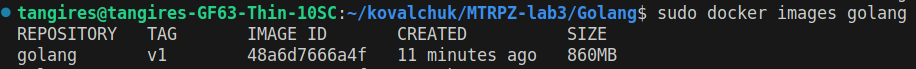
**Висновок:**

При порівняні двох використовуваних мною баз ( alpine та slim-bullseye) я побачила, що образ alpine більший за slim-bullseye в 2 рази та займає набагато більше часу при збірці. Це може залежати від скачування додаткових пакетів та ресурсів для коректної роботи. Я дійшла до висновку, що якщо нам потрібна швидка та ефективна робота образу, то кращим варіантом буде використання slim-bullseye, але якщо потрібні компоненти для роботи відсутні у цій базі, то будемо використовувати alpine.

1. **Golang**

**Збірка №1**

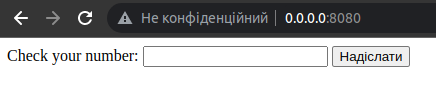
****

**Розмір:** 860MB

**Час:** 26,3 с

**Запускаємо:** sudo docker run –d –p 8080:8080 --name server --rm golang:v1

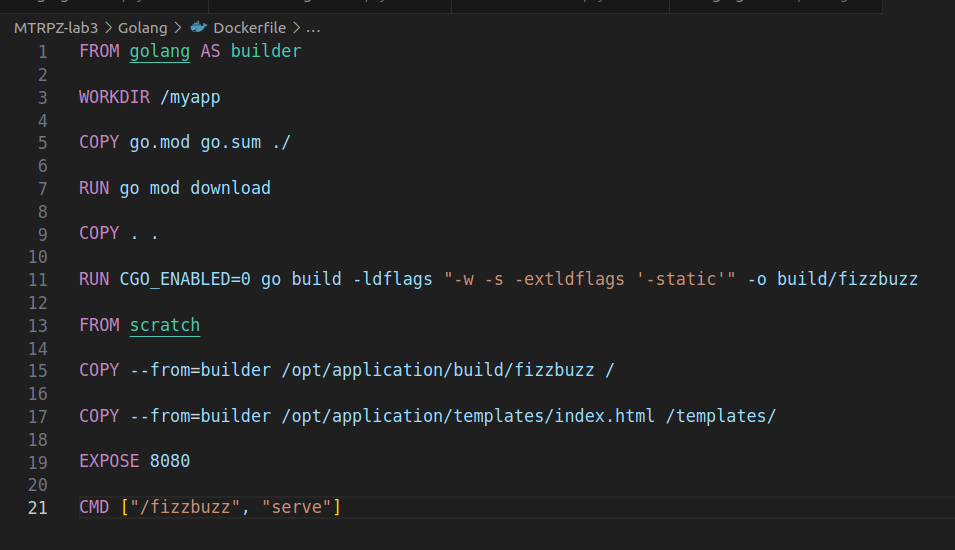
**Отримуємо:**

****

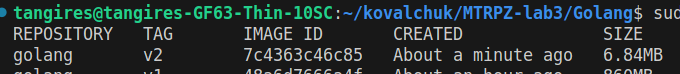
**Висновок:**

Після виконання успішної збірки я проаналізувала код та вирішила перевірити чи всі файли додані до образу необхідні для його успішної роботи та чи можна прибрати зайві файли для зменшення розмір образу, що вплине на ефективність його використання.

**Збірка №2**

****

**Розмір:** 6,84MB

****

**Час:** 8,3 с

**Запускаємо:** sudo docker run –d –p 8080:8080 --name server --rm golang:v2

Отримуємо:

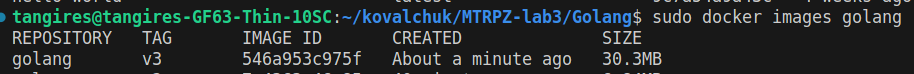


**Висновок:**

Використання багатоетапної збірки образу на базі `Scratch` демонструє його ефективність, оскільки цей підхід дозволяє створювати контейнер з програмою, що не містить непотрібних файлів або залежностей. Такий образ має дуже маленький розмір і містить лише необхідні складові для виконання програми, що забезпечує ефективність та безпеку роботи контейнера.

**Збірка №3**

**Розмір:** 30.3MB



**Час:** 9,7 с

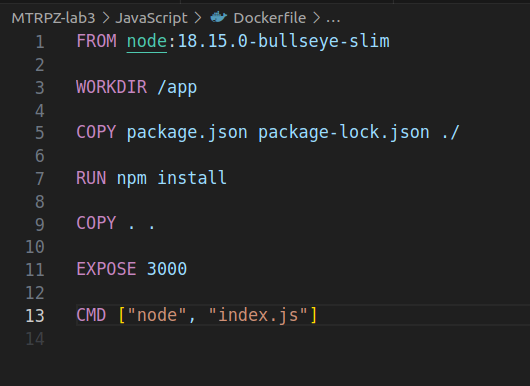
**Запускаємо:** sudo docker run –d –p 8080:8080 --name server --rm golang:v3

**Отримуємо:**

**Висновок:**

Спочатку я використала базовий образ scratch, бо з ним можна створити легкі за розміром та швидкі за часом збірки образи, потім обрала образ gcr.io/distroless/base, який має додактові компоненти, тому трохи переважає у розмірах та збірка зайняла більше часу. Порівнюючи можу сказати, що друга версія є більш простою у використанні, але вибір між цими двома підходами залежить від конкретних вимог у проекті.

1. **JavaScript**

****

**Розмір:** 253MB

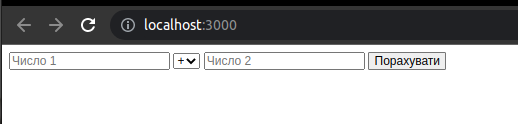
****

**Час:** 1.7 секунди

**Запускаємо:** sudo docker run –p 3000:3000 my-web-app:v1

**Отримуємо:**





**Висновок по лабораторній роботі:**

При проведенні лабораторної роботи я займалась збіркою docker-образів проектів на різних мовах програмування ( Python, Golang та JavaScript ). Проаналізувавши усі результати збірок, можу зробити наступні висновки:

1) Досліджуючи розмір образу я помітила, що він досить сильно змінюється від використання залежностей, тому при використанні менших базових образів ми можемо суттєво зменшити у розмірах кінцевий образ.

2) За часом збірки образи змінюються від багатьох факторів, наприклад: стартовий розмір проекту, наявність залежностей, налаштування Dockerfile та ін. Тому найбільш ефективним рішенням буде використання кешування, для перевикористання попередньо зібраних шарів.

3) Вибір базового образу перш за все впливає на розмір образу, тому використання менших за розміром базових образів вважаю більш ефективним, адже вони зменшують розмір кінцевого образу та часу збірки.

Отже, у результаті виконання цієї роботи я навчилась працювати з docker-образами та краще розуміти процес їх збірки, вплив на розмір проекту та час збирання. Використовуючи різні мови програмування я змогла зробити порівняння та висновки, які допомогли мені зрозуміти як краще приймати та обґрунтовувати свої рішення щодо впровадження ефективності в процес оптимізації збірки docker-образів для майбутніх проектів.