Лабораторна робота №11

Основи роботи з Docker-контейнерами

Мета: освоїти основні засоби та прийоми роботи з віртуальними контейнерами на базі Docker (на прикладі Docker Desktop).

Інструменти: Docker Desktop (на базі Windows, Linux або macOS), робоче Інтернет-з’єднання.

Під час цієї лабораторної роботи ви:

* ознайомитеся з особливостями технології віртуальних контейнерів на базі просторів імен на прикладі технології Docker;

* навчитеся:

- створювати віртуальний контейнер на базі готового образу Docker Images;

- використовувати базові команди для роботи з Docker-контейнерами (`run`, `ps`, `exec`, `stop`, `logs`, `exec`, `restart`, `rm`, `build`);

- запускати простий тестовий вебсервер на базі Nginx у Docker-контейнері;

- покращувати основні безпекові налаштування Docker-контейнера.

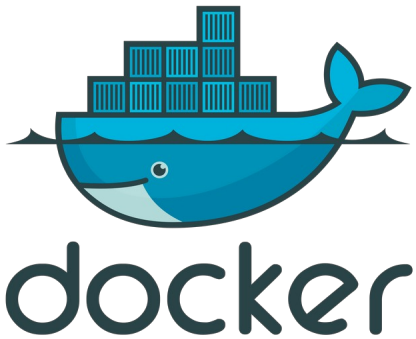
Теоретичні відомості та хід роботи

ВІРТУАЛЬНІ КОНТЕЙНЕРИ DOCKER

Docker є одним із найпопулярніших засобів віртуалізації на базі технології віртуальних контейнерів (також відомої як контейнеризація).

Docker використовується для:

- розробки та розгортання програмного забезпечення у заздалегідь налаштованому віртуалізованому середовищі;
- автоматизації тестування та CI/CD (безперервної інтеграції / безперервної доставки);
- розгортання середовищ для тестування застосунків на вразливості;



- побудови систем, що складаються з багатьох мікросервісів (кожний – у своєму контейнері);
- перенесення застосунків у хмарні середовища, на IoT-пристрої тощо.

На відміну від віртуальних машин, на кожній з яких працює окрема операційна система з окремим ядром, віртуальні контейнери використовуються *спільне ядро операційної системи*. Це ядро Linux або іншої Unix-подібної операційної системи. Водночас, завдяки допоміжним технологіям віртуалізації, контейнери запускають і на інших основних ОС, зокрема на Windows та macOS.

Віртуальний контейнер має:

- окрему файлову систему;
- окреме оточення (процеси, мережні інтерфейси, хостнейм);
- окремі системні ресурси (оперативна пам’ять, віртуальні ядра ЦП тощо).

Серед інших прикладів засобів віртуалізації на базі віртуальних контейнерів – FreeBSD-jail, LXC, OpenVZ, Parallels Virtuozzo, Linux-VServer. Ці технології по-різному вирішують питання відокремлення віртуального контейнера від решти системи (простори імен, мітки, зони тощо). Docker реалізує це завдання через *простори імен* (namespaces).

Віртуальні контейнери можна швидко створити і швидко видалити, коли вони перестануть бути потрібні. Під час роботи вони споживають менше ресурсів, ніж віртуальні машини. Це головні переваги віртуальних контейнерів, завдяки яким дана технологія й набула настільки значного поширення.

Водночас, віртуальні контейнери мають слабшу ізоляцію, ніж у віртуальних машин, тому, наприклад, у кібербезпеці вони часто використовуються разом віртуальними машинами. Та й загалом, застосування поєднання віртуальних машин та віртуальних контейнерів дуже розповсюджене. Скажімо, якщо основна ОС архітектурно не підтримує контейнери, то спершу створюється віртуальна машина (наприклад, полегшена ВМ з Linux для роботи Docker у Windows), а вже всередині цієї віртуальної машини запускаються віртуальні контейнери. І навпаки, можливий запуск віртуальної машини у віртуальному контейнері з метою створення зручного середовища для цієї машини (наприклад, WinBoat[1]).

Для запуску Docker-контейнерів на Linux-сервері використовують Docker Engine, а для настільного запуску Docker-контейнерів – Docker Desktop. У цій лабораторній роботі ми використовуватимемо Docker Desktop.

[1] Віртуалізаційне ПЗ WinBoat (https://winboat.app/) встановлюється на ОС Linux, поміщає у Docker-контейнер віртуальну машину на базі KVM та QEMU і на цій віртуальній машині запускає гостьову ОС Windows 11. Такий підхід забезпечує менше споживання ресурсів гостьовою ОС, охайне “пакування” віртуальної машини з Windows в єдиний QEMU-процес у віртуальному контейнері з усіма необхідними залежностями, віртуальними мережними інтерфейсами тощо, а також автоматизацію встановлення Windows.

ВСТАНОВЛЕННЯ DOCKER DESKTOP

Docker Desktop є настільним середовищем для роботи з Docker-контейнерами. Docker Desktop працює як на Linux (рідній для Docker), так і на інших операційних системах (Windows, macOS). Версію під свою ОС можна обрати на офіційному сайті: https://docs.docker.com/get-started/get-docker/

Для основної ОС Windows[2] на процесорах Intel/AMD у більшості випадків підійде редакція ***WSL 2, x86_64***. Дана редакція використовує підсистему Windows для Linux, версію 2 (Windows Subsystem for Linux 2). WSL 2 – це технологія віртуалізації на базі адаптованого ядра Linux, яке працює на спеціальній полегшеній віртуальній машині. З рекомендаціями щодо інших редакцій Docker Desktop для Windows можна ознайомитися в документації [1].

Docker Desktop можна встановити *для поточного користувача (per-user)* або *для всіх користувачів (all users)*. У більшості випадків встановлення лише для поточного користувача простіше (не потребує адміністративних прав) і безпечніше (менша поверхня атаки).

Для встановлення Docker Desktop на основну ОС Windows виконайте наступну послідовність кроків.

Крок 1. Завантажте інсталяційний файл Docker Desktop з офіційного сайту: https://docs.docker.com/desktop/release-notes/

Крок 2. Запустіть інсталяційний файл `Docker Desktop Installer.exe`. Для встановлення для поточного користувача оберіть `Per-user`.

Крок 3. На етапі вибору бекенду вкажіть `WSL 2`.

Крок 4. Дотримуючись інструкцій майстра встановлення, авторизуйте інсталятор та продовжте встановлення. Коли встановлення успішно завершиться, натисніть `Close`.

В основній ОС Linux середовище Docker Desktop використовує демон systemd, гіпервізор KVM та повний емулятор QEMU. Тобто Docker Desktop запускає свої контейнери на спеціальній віртуальній машині, спільній для всіх контейнерів, і саме для роботи цієї віртуальної машини потрібні KVM та QEMU. А от у разі використання лише базового компоненту Docker, Docker Engine (як у серверних сценаріях), спеціальна віртуальна машина не потрібна.

> ***Примітка.*** Docker Desktop працює лише на Linux, встановленій на фізичний комп'ютер, і не підійде для Linux на віртуальній машині (бо знадобилося б запустити віртуальну машину всередині віртуальної машини, а Docker Desktop наразі не підтримує роботу у середовищах на базі вкладної віртуалізації). Тож, якщо ви хотіли б спробувати Docker на віртуальній машині з Linux, скористайтеся Docker Engine без Docker Desktop.

[2] Тут ідеться *лише* про настільні версії Windows (10/11). Робота Docker на Windows Server відрізняється на різних версіях і детальніше описана у документації: https://learn.microsoft.com/uk-ua/virtualization/windowscontainers/quick-start/set-up-environment?tabs=dockerce

Хід встановлення Docker Desktop на Linux різниться залежно від дистрибутиву. У цій лабораторній роботі ми розглянемо інсталяцію Docker Desktop на Ubuntu [4]. Інструкції та інсталяційні файли для інших дистрибутивів Linux (зокрема для Debian, RHEL, Fedora, Arch) шукайте на офіційному сайті Docker: https://docs.docker.com/desktop/setup/install/linux/.

Для встановлення Docker Desktop на основну ОС Linux виконайте наступну послідовність кроків.

Крок 1. Якщо ви використовуєте графічну оболонку, відмінну від GNOME (KDE, LXDE, Xfce тощо), то на початку необхідно встановити пакунок `gnome-terminal`:

```
sudo apt install gnome-terminal
```

Крок 2. Підключіть репозиторій Docker:

```
# Додаємо GPG-ключ для підтвердження автентичності репозиторію:
sudo apt update
sudo apt install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Додаємо репозиторій до переліку джерел пакунків Apt:
sudo tee /etc/apt/sources.list.d/docker.sources <<EOF
Types: deb
URIs: https://download.docker.com/linux/ubuntu
Suites: $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}")
Components: stable
Architectures: $(dpkg --print-architecture)
Signed-By: /etc/apt/keyrings/docker.asc
EOF

sudo apt update
```

Навіщо: підключення репозиторіїв допоможе apt коректно підтягнути всі необхідні залежності під час встановлення пакунку з Docker Desktop на кроці 4.

Крок 3. Завантажте deb-файл пакунку з Docker Desktop з офіційного сайту (https://docs.docker.com/desktop/release-notes/, оберіть свіжий реліз для Debian).

Зробіть поточним каталог, у який завантажили deb-файл. Наприклад:

```
cd Завантажене
```

Крок 4. Встановіть пакунок з Docker Desktop:

```
sudo apt install ./docker-desktop-amd64.deb
```

В основній ОС macOS Docker Desktop використовує для запуску віртуальних контейнерів віртуальну машину з Linux, яка працює на базі технології Apple Virtualization framework (для процесорів Apple Silicon) або HyperKit (для процесорів Intel).

Крок 1. Завантажте інсталяційний dmg-файл Docker Desktop для процесора Apple Silicon або Intel: https://docs.docker.com/desktop/release-notes/

Крок 2. Відкрийте файл `Docker.dmg` та перетягніть його в розділ `Applications`.

Крок 3. Відкрийте Docker з розділу `Applications` та погодьтеся з ліцензійними умовами для першого запуску Docker Desktop.

Крок 4 (додатковий, лише для процесорів Apple Silicon). Для комфортної роботи з Docker Desktop на macOS рекомендовано також встановити транслятор коду Rosetta 2:

```
softwareupdate --install-rosetta
```

ПЕРШИЙ ЗАПУСК DOCKER DESKTOP

Після встановлення Docker Desktop доступний у головному меню вашої операційної системи.

Під час першого запуску вам буде запропоновано переглянути ліцензійні умови (Рис. 11.1). Для продовження роботи з Docker потрібно погодитися з цими умовами (якщо ви ще проходили цей крок на етапі встановлення).

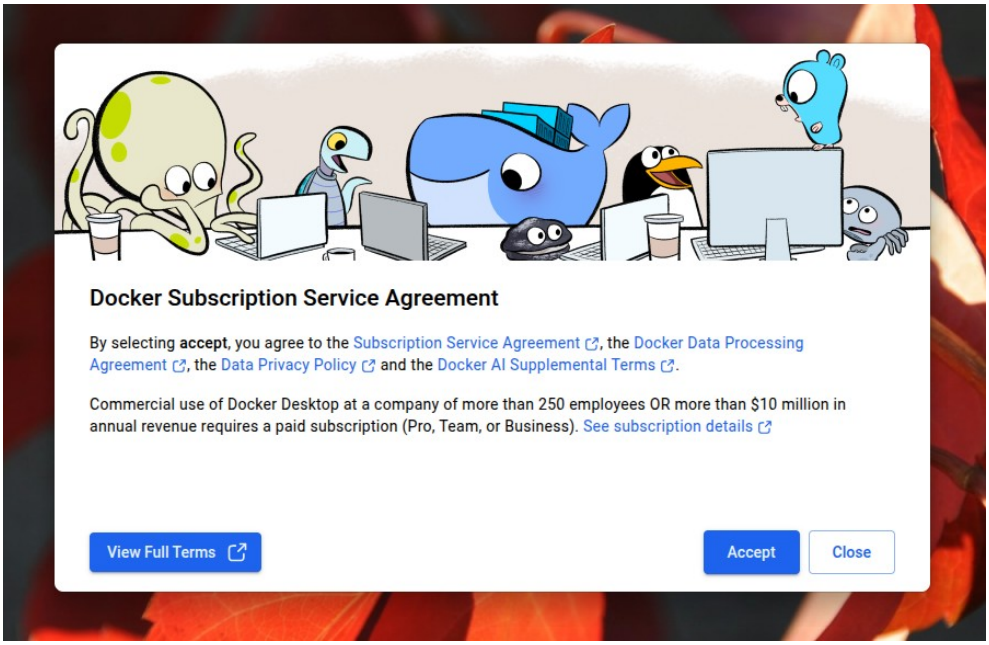


Рис. 11.1. Діалогове вікно погодження з ліцензійними умовами перед використанням Docker

У наступному діалоговому вікні можна увійти в обліковий запис Docker або створити такий обліковий запис (Рис. 11.2). Вхід в обліковий запис надає доступ до власного репозиторію образів Docker-контейнерів та, згідно з документацією, підвищує швидкість завантаження образів з репозиторію. Але цей крок не є обов'язковим, і його можна пропустити, натиснувши `Skip`.

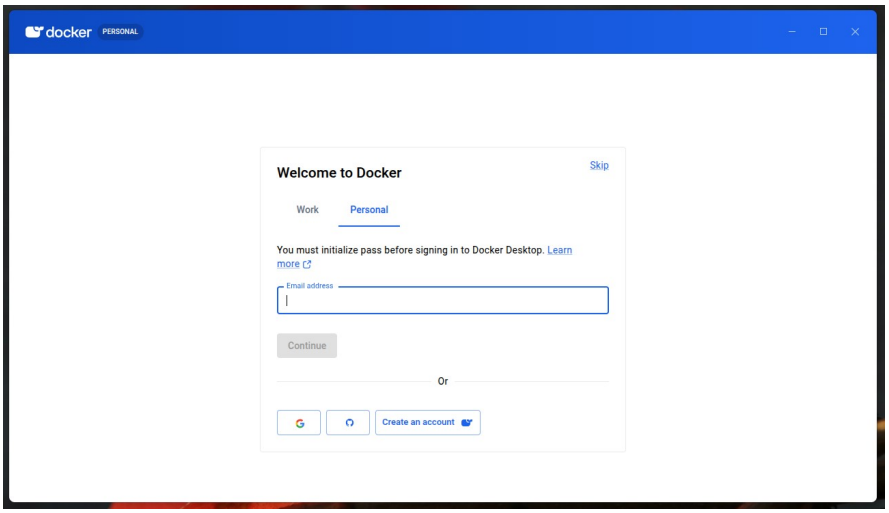


Рис. 11.2. Вікно входу в обліковий запис Docker (необов’язковий крок)

Далі ви побачите головне вікно програми Docker Desktop (Рис. 11.3).

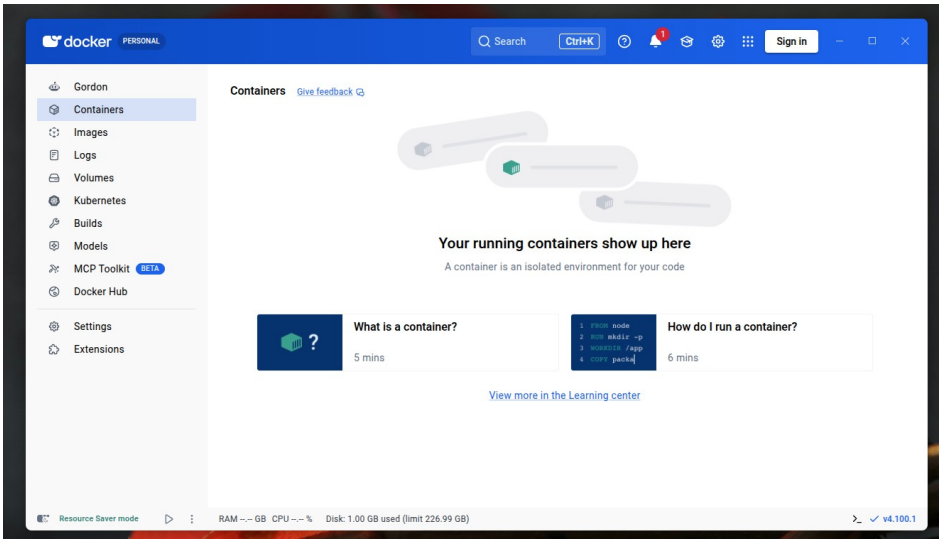


Рис. 11.3. Головне вікно програми Docker Desktop одразу після встановлення

У лівому нижньому куті головного вікна відображається стан компонента Docker Engine (Рис. 11.4). Переконайтеся, що це стан *Engine running*, якщо це не так – натисніть на трикутник поряд з індикатором, щоб запустити (може знадобитися трохи зачекати, доки стан зміниться).

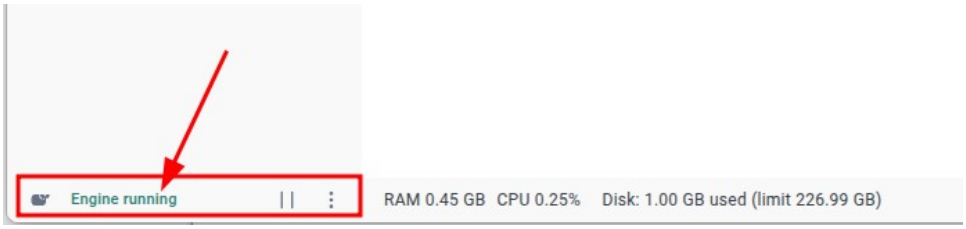


Рис. 11.4. Індикатор стану (*Engine running*)

У Docker Desktop є власна панель терміналу. У цій роботі ми будемо використовувати передусім команди, додатково спостерігаючи ефект від них в графічному інтерфейсі. Щоб активувати панель терміналу, у правому нижньому куті головного вікна Docker Desktop натисніть на значок >_ , а тоді на кнопку Enable (Рис. 11.5).

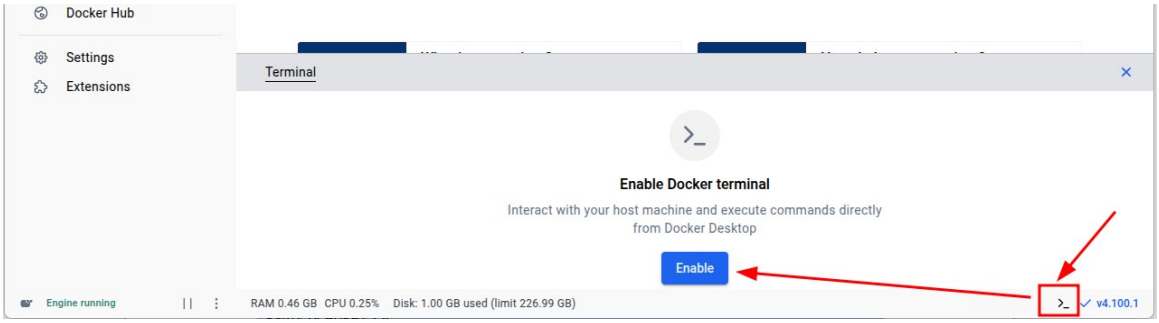


Рис. 11.5. Активування панелі терміналу

Приклад вигляду панелі терміналу після активування показано на Рис. 11.6. Термінал на скриншоті зроблено в Docker Desktop на основній ОС Ubuntu Linux, на Windows та macOS його вигляд може трохи відрізнятися.

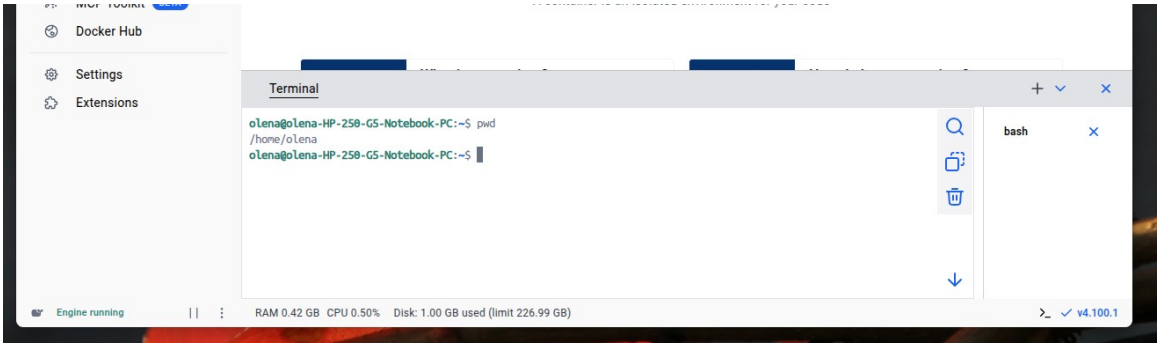


Рис. 11.6. Приклад панелі терміналу у Docker Desktop

Вкладка *Containers*. Docker Desktop дозволяє виконувати більшість операцій з контейнерами через графічний інтерфейс, але на етапі навчання буде корисно виконувати ці операції через панель терміналу, паралельно відстежуючи результати в графічному інтерфейсі. Наприклад, на Рис. 11.7 контейнер було запущено через панель терміналу (нижня половина скриншоту), і водночас його видно в графічному інтерфейсі (верхня половина скриншоту).

Вкладка *Images*. Аналогічно, на Рис. 11.8 показано, як відомості про попередньо завантажений образ *nginx:latest* відображаються у графічному інтерфейсі Docker Desktop (верхня половина скриншоту) та через панель терміналу (нижня половина скриншоту).

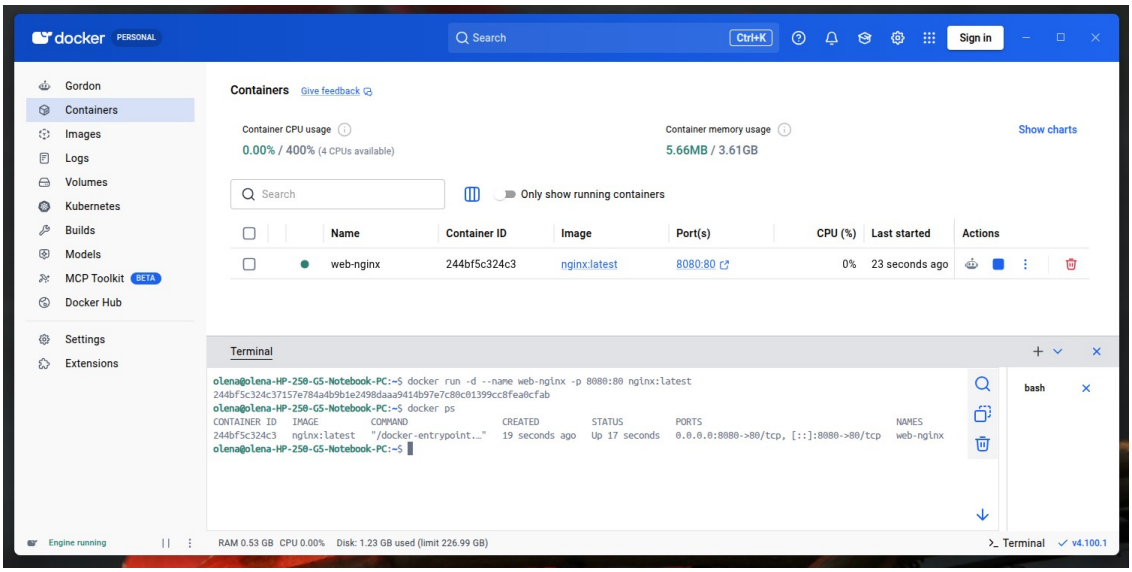


Рис. 11.7. Головне вікно Docker Desktop з активною вкладкою *Containers* та запущеним віртуальним контейнером *nginx*

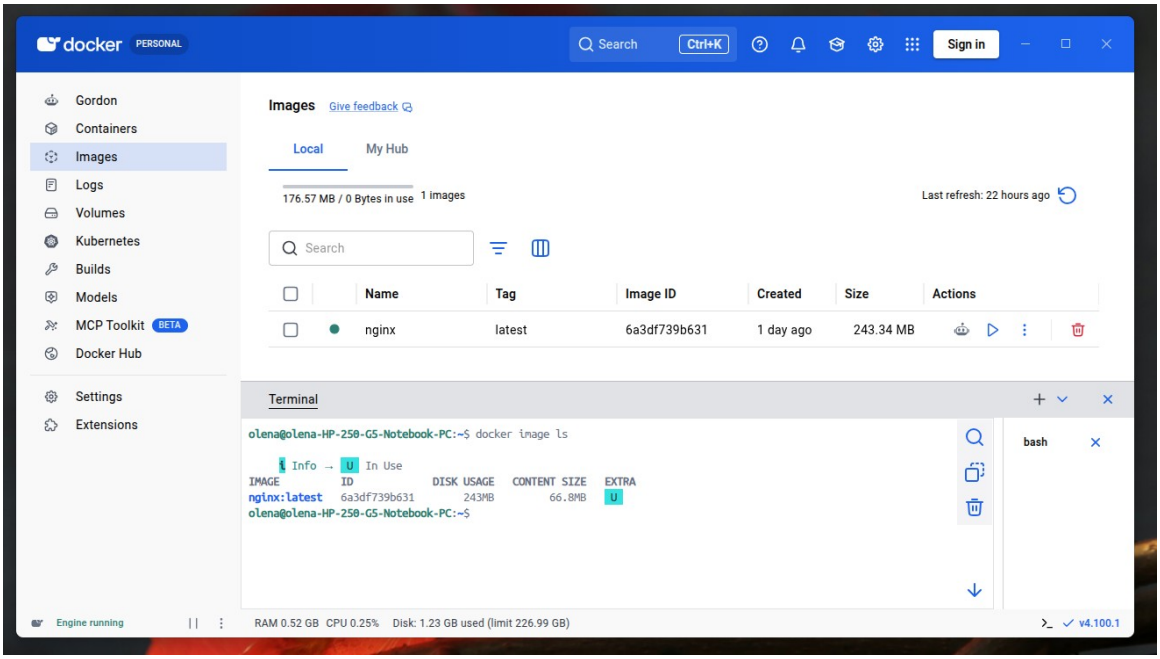


Рис. 11.8. Головне вікно Docker Desktop з активною вкладкою *Images* > *Local* та завантаженим раніше образом *nginx:latest*

☝️ Саме такий підхід ми й будемо використовувати далі у цій роботі:

- >_ працюємо засобами ***терміналу Docker Desktop***;
- 🗓️ відстежуємо результати своїх дій у ***GUI Docker Desktop***.

Завдання №11.1. Підготовка середовища Docker Desktop до роботи

1. Використовуючи наведені вище інструкції, встановіть Docker Desktop на вашу основну ОС.

2. Виконайте перший запуск середовища Docker Desktop. Переконайтеся, що Docker Engine запущено (за потреби запустіть). Активуйте панель терміналу Docker Desktop.

3. Для файлів даної лабораторної роботи створіть окремий каталог ***docker-lab13-surname-var*** у вашій основній ОС та зробіть його поточним у терміналі Docker Desktop (*surname* – ваше прізвище, *var* – ваш варіант). Цей поточний каталог має відображатися у запрошенні на ваших скриншотах за наступними завданнями.

Звіт має містити скриншоти основних кроків та відповідні пояснення.

БАЗОВІ КОМАНДИ ДЛЯ РОБОТИ З DOCKER-КОНТЕЙНЕРАМИ

Docker-контейнери запускають на основі заздалегідь підготовлених образів (images). Можна використовувати власні образи, але Docker Hub (https://hub.docker.com/) пропонує велику кількість готових образів, які можна доналаштувати відповідно до власних потреб. *Офіційно перевірені образи* належать до категорій:

- Docker official images (офіційні образи, які мають регулярну підтримку);
- Docker Hardened Images (образи з покращеним захистом та регулярною підтримкою);
- verified publishers (образи від перевірених компаній, також з регулярною підтримкою);
- sponsored open source (образи з opensource-рішеннями, спонсоровані Docker, зазвичай також з регулярною підтримкою).

У цій лабораторній роботі ми використовуватимемо образи з категорії Docker official images, що є рекомендованою практикою для початківців.

☆ *Команда `docker pull`*

Завантажити образ віртуального контейнера можна за допомогою команди *docker pull*:

```
docker pull ім'я_образу
```

Наприклад:

```
docker pull nginx:latest
```

У разі, якщо ім'я образу було вказане правильно, і такий образ доступний з Docker Hub, його буде завантажено на диск.

☆ ***Команда `docker image`***

Дана команда призначена для роботи із завантаженими образами контейнерів. Їх можна перебудовувати, імпортувати, видаляти, досліджувати відомості про них та виконувати інші дії.

Для перегляду основних відомостей про завантажені образи контейнерів (ідентифікатор образу, розмір на диску, обсяг вмісту контейнера, чи образ зараз використовується тощо) можна скористатися наступною командою:

```
docker image ls
```

☆ ***Команда `docker run`***

Для запуску контейнера командою *docker run* достатньо вказати ім'я відповідного образу. Водночас, вказавши більше параметрів, можна отримати кращий досвід взаємодії з контейнером:

```
docker run -d --name ім'я_контейнера
-p порт_хоста:порт_контейнера ім'я_образу
```

Даний контейнер буде запущено у фоновому режимі (*-d*, *detached*), щоб після запуску контейнера термінал і надалі був доступний для введення нових команд. Також ім'я контейнера задано вручну (`--name ім'я_контейнера`), оскільки інакше йому буде присвоєне випадково згенероване ім'я. Номери портів хоста та контейнера (`-p порт_хоста:порт_контейнера`), через які хост і контейнер будуть взаємодіяти, також може знадобитися вказати вручну, особливо якщо вони відрізняються від стандартних.

☝ **Дані, які зберігатимуться безпосередньо у запущеному контейнері, видалятимуться під час зупинки цього контейнера.** Тому якщо потрібні дані, які зберігатимуться постійно, незалежно від того, запущений зараз контейнер чи ні, їх потрібно розмістити поза межами контейнера і підключити до контейнера як віртуальний *том (volume)*.

Монтування тому під час запуску контейнера здійснюється за допомогою параметра *-v*. Скажімо, якщо йдеться про ресурс (файл чи каталог) у локальній файловій системі хоста, то це може виглядати так:

```
docker run -d --name ім'я_контейнера -p порт_хоста:порт_контейнера
-v "шлях_до_ресурсу_у_файловій_системі_хоста:точка_монтування_у_файловій_системі_контейнера:права_доступу"
ім'я_образу
```

Наприклад:

```
docker run -d --name web-nginx -p 8080:80 -v
"$(pwd)/index.html:/usr/share/nginx/html/index.html:ro" nginx:latest
```

де `-d` – фоновий режим запуску контейнера;

`--name web-nginx` – ім'я контейнера (`web-nginx`);

`-p 8080:80` – порт хоста (`8080`) та порт контейнера (`80`);

`-v "./index.html:/usr/share/nginx/html/index.html:ro"` – монтування файлу `index.html` як віртуального тому, шлях до якого у локальній системі хоста буде `./index.html` (крапка відповідає поточному каталогу), а у

файловій системі він буде доступний з точки монтування `/usr/share/nginx/html/index.html`, причому процесу контейнера дозволено лише читання цього файлу (`ro`);

`nginx:latest` – ім'я образу (image), на основі якого розгортається контейнер.

Більше про віртуальні томи Docker (Docker volumes) є у документації Docker: https://docs.docker.com/engine/storage/volumes/.

☆ *Команда `docker ps`*

Для перегляду інформації про наявні у системі контейнери виконайте команду *docker ps*. Використана без параметрів, вона виведе відомості лише про запущені контейнери:

```
docker ps
```

Якщо з потрібно побачити відомості про всі контейнери (запущені й ні) – додайте параметр *-a*:

```
docker ps -a
```

☆ *Команда `docker logs`*

Для перегляд повідомлень аудиту про роботу віртуального контейнера скористайтеся командою *docker logs:*

```
docker logs ім'я_контейнера
```

☆ *Команда `docker exec`*

Щоб гостьова операційна система всередині контейнера виконала ту чи іншу команду, помістіть цю команду після команди *docker exec*, вказавши ім'я потрібного контейнера.

Наприклад, для одержання відомостей про Linux-систему, яка працює всередині контейнера, скористайтеся командою:

```
docker exec ім'я_контейнера cat /etc/os-release
```

Для виведення інформації про всі процеси, запущені всередині контейнера, може підійти команда:

```
docker exec ім'я_контейнера ps aux
```

Водночас обов'язково враховуйте, що команду має підтримувати гостьова ОС, яка працює всередині контейнера. Інакше ви одержите повідомлення про помилку.

☆ *Команда `docker stop`*

Щоб зупинити віртуальний контейнер, використовується команда *docker stop*:

```
docker stop ім'я_контейнера
```

Важливе зауваження. Зупинка контейнера означає знищення всіх даних, які зберігаються всередині контейнера і які не включені у версію образу, на основі якого цей контейнер розгортається. Тому якщо застосунок, що працює всередині контейнера, наприклад, використовує базу даних, то її потрібно зберігати поза контейнером. Це можна зробити на основі віртуальних томів Docker.

☆ *Команда `docker restart`*

Зупинений контейнер не вийде знову запустити командою *docker run*, вони використовується лише для першого запуску контейнера. Для повторного запуску зупиненого контейнера використовуйте *docker restart*:

```
docker restart ім'я_контейнера
```

☆ *Команда `docker rm`*

Вилучення віртуального контейнера здійснюється командою *docker rm*:

```
docker rm ім'я_контейнера
```

☆ *Опція `--help` та автодоповнення клавішею Tab*

Довідкову інформацію за командами Docker можна одержати за допомогою опції *--help*. Наприклад, для одержання загальної довідки за всіма командами введіть:

```
docker --help
```

Також можна вивести довідку за конкретною командою. Наприклад, довідку за командою docker run можна одержати так:

```
docker run --help
```

У терміналі Docker працює й **автодоповнення клавішею Tab** (для команд, назв параметрів, імен образів, контейнерів тощо).

Наприклад, щоб не писати вручну ім'я контейнера, який потрібно зупинити, виконайте таку послідовність дій.

1. Почніть писати команду: `docker stop`

2. Поставте *пробіл* після `stop` – так інтерпретатор «зрозуміє», що потрібно не доповнити символи, які йдуть далі після «*stop*», а запропонувати параметр, який іде після пробілу.

3. Натисніть *Tab*. Має з'явитися перша пропозиція з ім'ям контейнера. Якщо ви хочете зупинити інший контейнер, продовжуйте натискати *Tab*, доки не одержите потрібне ім'я. Якщо у вас багато контейнерів, можна спершу ввести кілька перших символів імені потрібного контейнера, щоб обмежити коло пошуку.

4. Побачивши потрібне ім'я контейнера, натисніть *Enter*. Запуститься команда зупинки контейнера.

Завдання №11.2. Запуск першого контейнера (Nginx)

Працюємо засобами ***терміналу Docker Desktop*.**

Відстежуємо результати своїх дій у ***графічному інтерфейсі Docker Desktop***

1. Завантажте* образ *nginx:latest* з Docker Hub.

* *Примітка.* Технічно цей крок не є обов'язковим, оскільки команда *docker run* автоматично підвантажує образ контейнера з Docker Hub, якщо такого ще немає в користувача. Але нам цей крок допоможе переконатися, що ми працюємо з

найбільш актуальною версією даного образу (*latest*).

2. Перегляньте відомості про завантажений образ *nginx:latest*. Скільки місця він займає на диску? Який обсяг вмісту контейнера?

3. Запустіть віртуальний контейнер на основі образу *nginx:latest* з наступними параметрами:

- ☐ ім’я контейнера – *web-nginx*;
- ☐ порт хоста – 8080;
- ☐ порт контейнера – 80;
- ☐ режим запуску контейнера – фоновий.

4. Переконайтеся, що контейнер *web-nginx* працює (команда *docker ps*).

5. Відкрийте веббраузер і перейдіть за адресою `http://localhost:8080`. Має з'явитися стандартна сторінка «Welcome to nginx!».

6. Перегляньте журнали веб-сервера nginx (команда *docker logs*). Знайдіть у журналах запис, що стосується запиту від вашого браузера, який ви робили у п. 5.

7. Дослідіть контейнер *web-nginx*:

- ☐ з’ясуйте дистрибутив Linux та номер релізу;
- ☐ виведіть ім’я поточного каталогу;
- ☐ визначте користувача, від імені якого виконуються команди в контейнері;
- ☐ перевірте, чи працюють команди id, ps, top.

8. Зупиніть контейнер *web-nginx* і переконайтеся, що він справді зупинився.

9. Повторно запустіть контейнер *web-nginx* і перевірте, чи його запущено.

10. Знову зупиніть контейнер *web-nginx* і цього разу видаліть його. Переконайтеся, що контейнера немає (і серед зупинених теж).

Звіт має містити скриншоти основних кроків та відповідні пояснення.

Завдання №11.3. Створення власної вебсторінки та монтування її в контейнер

Працюємо засобами ***терміналу Docker Desktop*.**

Відстежуємо результати своїх дій у ***графічному інтерфейсі Docker Desktop***

1. У каталозі *docker-lab13-surname-var* створіть файл *index.html* з кодуванням UTF-8 та наступним вмістом (підставивши власні прізвище та ім’я):

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Лабораторна робота №13</title>
```

```
    </head>
    <body>
        <h1>Операційні системи. Лабораторна робота №11.</h1>
        <p>Виконав / виконала: Прізвище, Ім'я</p>
    </body>
    </html>
```

2. Запустіть новий контейнер на базі образу *nginx:latest*.

Характеристики нового контейнера:

☐ *ім'я* можна використати таке саме, як і для попереднього контейнера (*web-nginx*), адже попередній контейнер ви раніше видалили;

☐ *порти* також лишаємо такі самі, як у попереднього контейнера (8080:80);

☐ *вебсторінку index.html* контейнер має брати не власну стандартну, а створену вами у п. 1 (скористайтеся монтуванням віртуального тому, параметр *-v*).

3. Оновіть сторінку `http://localhost:8080` у браузері та зробіть скриншот результату. Вигляд і вміст сторінки має відповідати файлу *index.html*, створеному у п. 1.

Дайте відповідь на питання: З якою метою вебсторінку index.html винесено за межі контейнера і розміщено у файловій системі локального хоста? (Чому недоцільно розміщувати її безпосередньо всередині контейнера?)

4. Зупиніть контейнер *web-nginx*. Вилучіть цей контейнер.

<u>Звіт</u> має містити скриншоти основних кроків та відповідні пояснення, а також короткі текстові відповіді на поставлені питання.

МОДИФІКАЦІЯ DOCKER-КОНТЕЙНЕРА ЗА ДОПОМОГОЮ ФАЙЛУ DOCKERFILE

Використання *Dockerfile* дозволяє додати до готового образу власні налаштування (параметри запуску, застосунок, який буде працювати всередині контейнера, команди, які потрібно виконати після запуску контейнера тощо).

Загальну структуру файлу *Dockerfile* описано у табл. 13.1.

Таблиця 13.1

Загальна структура файлу *Dockerfile*

Інструкція	*Опис*
`FROM` *`ім'я_образу`*	Визначає основу вашого образу

Інструкція	*Опис*
`RUN` *`команда`*	Виконує вказані команди на новому прошарку поверх готового образу.
`WORKDIR` *`шлях_до_каталогу`*	Встановлює поточний (робочий) каталог, у якому виконуються всі інші інструкції, задані у файлі *Dockerfile*.
`COPY` *`звідки_копіювати куди_копіювати`*	Копіює нові файли чи каталоги з каталогу в файловій системі локального хоста у файлову систему контейнера.
`CMD` *`команди`*	Програми (команди), які буде запущено одразу після старту контейнера. Команд можна вказати декілька: `["команда_1", "команда_2"]`.

☆ *Команда `docker build`*

Ця команда дозволяє збудувати власний образ Docker-контейнера на основі готового образу. Налаштування, які потрібно використати під час побудови власного образу, потрібно заздалегідь задати у файлі `Dockerfile`. Загальний вигляд побудови образу може виглядати так:

```
docker build -t ім'я_нового_образу шлях_до_файлів
```

Результатом роботи команди буде новий образ з ім'ям *`ім'я_нового_образу`*, побудований на основі файлу Dockerfile, розміщеного у каталозі *`шлях_до_файлів`*.

Наприклад:

```
docker build -t sysinfo-app:v1 .
```

Тут результатом роботи команди стане образ `sysinfo-app:v1`, а файл *Dockerfile* братиметься з поточного каталогу (.). Якщо цей файл має стандартне ім'я (*Dockerfile*), то ім'я вказувати не обов'язково.

Завдання №11.4. Створення та запуск індикатора ізоляції контейнера

Працюємо засобами ***терміналу Docker Desktop.***

Відстежуємо результати своїх дій у ***графічному інтерфейсі Docker Desktop***

Що ми будемо робити. У цьому завданні ми напишемо і запустимо у віртуальному контейнері простий застосунок мовою Python. Це завдання також допоможе нам дослідити ізольованість контейнера від основної операційної системи, оскільки застосунок виводитиме відомості про власне оточення гостьової ОС контейнера (хостнейм, віртуальну IP-адресу, ідентифікатори процесу).

1. У каталозі *docker-lab13-surname-var* створіть файл *app.py* з наступним вмістом:

```
    import http.server
    import socketserver
    import socket
    import os

    PORT = 8080

    class InfoHandler(http.server.SimpleHTTPRequestHandler):
        def do_GET(self):
            hostname = socket.gethostname()
            ip_addr = socket.gethostbyname(hostname)
            uid = os.getuid()
            gid = os.getgid()

            html = f"""<!DOCTYPE html>
            <html lang="uk">
            <head><meta charset="UTF-8"><title>Docker Info</title></head>
            <body>
                <h2>Інформація про середовище контейнера</h2>
                <h2>Виконав / виконала: Прізвище, Ім’я</h2>
                <ul>
                    <li><b>Hostname:</b> {hostname}</li>
                    <li><b>Внутрішній IP:</b> {ip_addr}</li>
                    <li><b>UID / GID процесу python:</b> {uid} / {gid}</li>
                </ul>
            </body>
            </html>"""

            self.send_response(200)
            self.send_header("Content-type", "text/html; charset=utf-8")
            self.end_headers()
            self.wfile.write(html.encode("utf-8"))

    with socketserver.TCPServer(("", PORT), InfoHandler) as httpd:
        print(f"Сервер запущено на порті {PORT}")
        httpd.serve_forever()
```

2. Створіть `Dockerfile`. Цей файл не має розширення і використовується для додання до готового образу нового прошарку (*layer*) налаштувань.

Вміст `Dockerfile`:

```
FROM python:3.12-slim
WORKDIR /app
COPY app.py .
EXPOSE 8080
CMD ["python", "app.py"]
```

У нашому випадку, Dockerfile вказуватиме наступне:

- ☐ контейнер потрібно запустити на основі образу `python:3.12-slim`;
- ☐ поточним каталогом всередині контейнера буде каталог `/app`;
- ☐ всередину контейнера у поточний каталог (.) потрібно скопіювати файл `app.py`;
- ☐ зовнішнім портом контейнера (для взаємодії з хостом) буде порт `8080`;
- ☐ у контейнері треба послідовно виконати команди `python` та `app.py`.

3. Переконайтеся, що каталог каталозі *docker-lab13-surname-var* досі є поточним і містить Dockerfile. Побудуйте новий образ контейнера на основі цього файлу Dockerfile і назвіть цей образ *sysinfo-app:v1* (скористайтеся командою *docker build*).

Під час побудови образу спостерігайте за ходом операції у терміналі Docker.

4. Запустіть віртуальний контейнер на основі нового образу *sysinfo-app:v1*.

Характеристики нового контейнера:

- ☐ *ім'я* контейнера – *app-v1*;
- ☐ *порти* – 8080:8080.

5. Перегляньте інформацію, яку виводить застосунок, відкривши адресу http://localhost:8080 у браузері вашої основної ОС.

Дайте відповідь на питання: 1) Яку віртуальну (внутрішню) IP-адресу має віртуальний контейнер? Чи співпадає ця адреса з IP-адресою вашого комп'ютера, яку виводить основна ОС? 2) Який хостнейм має віртуальна гостьова ОС у контейнері? З якою іншою властивістю контейнера співпадає цей хостнейм? 3) Які *UID* та *GID* має процес *python*?

6. Поки що лишіть контейнер *app-v1* працювати. Якщо вам потрібно зробити перерву у виконанні лабораторної, зупиніть контейнер, але не видаляйте його.

Звіт має містити скриншоти основних кроків та відповідні пояснення, а також короткі текстові відповіді на запитання.

ОСНОВИ БЕЗПЕКИ DOCKER-КОНТЕЙНЕРІВ

За замовчуванням Docker запускає процеси в контейнері від імені користувача *root* (UID 0), причому простір імен користувачів контейнера не ізолюється від хоста: UID 0 у контейнері відповідає UID 0 в основній ОС. Ізоляція контейнера забезпечується лише іншими просторами імен (PID процесів, окремою файловою системою, віртуальними мережними інтерфейсами), а також базовими безпековими механізмами Linux.

Якщо у такому контейнеризованому вебзастосунку є вразливість (скажімо, вразливість до віддаленого виконання коду, remote code execution), то зловмисник зможе отримати доступ до командної оболонки з правами *root* всередині контейнера. Це створює значні безпекові ризики, як для застосунку всередині контейнера, так і для основної ОС, зокрема, ризик так званої *втечі з контейнера* (container escape).

Тому, відповідно до принципу мінімальних повноважень, вебзастосунок не повинен мати прав адміністратора, натомість запускатися від імені окремо створеного непривілейованого користувача.

Завдання №11.5. Покращення безпеки контейнеризованого застосунку

Працюємо засобами ***терміналу Docker Desktop*.**

Відстежуємо результати своїх дій у ***графічному інтерфейсі Docker Desktop***

Що ми будемо робити. У цьому завданні ми спершу дослідимо безпеку простого застосунку, створеного у завданні №11.4, а потім використаємо кілька прийомів для покращення його безпеки.

1. Переконайтеся, що контейнер *app-v1* досі запущений. Якщо його зупинено – запустіть знову (*docker restart*).

2. Уявіть, що дієте як потенційний зловмисник. Спробуйте здійснити запис у контейнер ззовні. За допомогою *docker exec* виконайте всередині контейнер команду `touch /app/hacked.txt`. Перевірте, чи вдалося створити цей файл, виконавши в контейнері команду `ls -la /app`.

Дайте відповідь на питання: Чи вдалося створити файл? Чому? Які ризики це несе? (див. теоретичні відомості перед цим завданням)

3. Зупиніть контейнер *app-v1* та видаліть його. Ми покращимо безпеку контейнеризованого застосунку і запустимо його знову.

4. Внесіть до файлу *Dockerfile*, з яким працювали раніше, наступні зміни:

Вміст оновленого `Dockerfile` (зміни виділено жовтим):

```
FROM python:3.12-slim

# Створення нових групи та користувача (непривілейованих):
# група appgroup з GID 10001;
# користувача appuser з UID 1001, основною групою appgroup
# та без можливості безпосереднього входу (-s /usr/sbin/nologin)
RUN groupadd -g 10001 appgroup && \
    useradd -u 10001 -g appgroup -s /usr/sbin/nologin -M appuser

WORKDIR /app

# Зміна власників файлу застосунку app.py на непривілейованих
COPY --chown=appuser:appgroup app.py .

# Перемикання на непривілейованого користувача
USER appuser

EXPOSE 8080
CMD ["python", "app.py"]
```

5. Збережіть зміни до файлу *Dockerfile*. На основі оновленого файлу Dockerfile перезберіть новий образ *sysinfo-app:secure*.

6. Запустіть контейнер під новим ім’ям *app-secure*, увімкнувши режим «read-only» для кореневої файлової системи та виділивши ізольовану область у пам'яті (tmpfs) для каталогу */tmp* із забороною виконання файлів (*noexec*).

Це можна зробити наступною командою (подано кількома рядками для більшої наочності):

```
docker run -d \
  --name app-secure \
  --read-only \
  --tmpfs /tmp:rw,noexec,nosuid,size=16m \
  -p 8080:8080 \
  sysinfo-app:secure
```

7. Перегляньте інформацію, яку виводить застосунок тепер, знову відкривши адресу http://localhost:8080 у браузері.

Дайте відповідь на питання: Порівняно з попередньою версією застосунку, яка інформація змінилася?

8. Виконайте всередині контейнера *app-secure* команду whoami (через *docker exec*).

Дайте відповідь на питання: Від імені якого користувача виконується процес контейнера?

9. Виконавши всередині команду *touch* (через *docker exec*), знову спробуйте створити у файловій системі контейнера файл */app/hacked.txt*.

Дайте відповідь на питання: Чи вдається створення файлу цього разу? Чому?

10. Перевірте запис у каталог */tmp* та запуск звідти виконуваного файлу (зазвичай запис у каталог */tmp* доступний для всіх користувачів, і це теж може створити безпекові ризики для нашого контейнеризованого застосунку).

Спочатку створіть у каталозі /tmp файл скрипта:

```
docker exec app-secure sh -c "echo 'Hello world' > /tmp/script.sh"
```

Наш скрипт безпечний і просто виводить на екран повідомлення «Hello world», але в такий самий спосіб могло бути додано шкідливий скрипт.

Дозвольте запуск скрипта:

```
docker exec app-secure sh -c "chmod u+x /tmp/script.sh"
```

А тоді спробуйте запустити скрипт:

```
docker exec app-secure sh -c "/tmp/script.sh"
```

Дайте відповідь на питання: Чи вдалося запустити скрипт. Чому?

11. Зупиніть та виділіть контейнер.

Звіт має містити скриншоти основних кроків та відповідні пояснення, а також короткі текстові відповіді на поставлені запитання.

Вимоги до звіту

Звіт має містити інформацію відповідно до завдань 13.1-13.5.

Корисні посилання

Завантаження Docker Desktop:

1. Get Docker: Docker Docs. URL: https://docs.docker.com/get-started/get-docker/

Встановлення Docker Desktop:

2. Install Docker Engine: Docker Docs. URL: https://docs.docker.com/engine/install/

3. Install Docker Desktop on Windows. URL: https://docs.docker.com/desktop/setup/install/windows-install

4. Install Docker Desktop on Ubuntu: https://docs.docker.com/desktop/setup/install/linux/ubuntu/

5. Install Docker Desktop on Mac: https://docs.docker.com/desktop/setup/install/mac-install/

Офіційна документація Docker:

6. Docker Docs. URL: https://docs.docker.com/