



Лабораторна робота №15

Тема: «Основи контейнерної віртуалізації програмного забезпечення»

Викладач: Олександр А. Блажко,

доцент кафедри ІС Одеської політехніки, blazhko@ieee.org

(для виконання під час *Online*-заняття груп AI221, AI222, AI223)

Мета роботи: придбання навичок із використання файлової віртуалізації на прикладі механізму *chroot* та віртуальної контейнеризації на прикладі програмного забезпечення *Docker*.

1 Теоретичні відомості

1.1 Файлова віртуалізація в *UNIX*-подібних ОС на основі команди *chroot*

В *Unix*-подібних ОС існує віртуалізація на основі механізму команди *chroot* (*change root*), який змінює кореневий каталог, дозволяючи різним програмам, запущеним зі своїми зміненими кореневими каталогами, мати доступ тільки до файлів з цих каталогів.

На рисунку 1 показано фрагмент ієрархії файлової системи *Ext* після виконання команди *chroot*, яка для якогось процесу змінює *root*-каталог з класичного */* на каталог */mnt/new root*, який раніше було створено у тимчасовму каталозі *mnt*

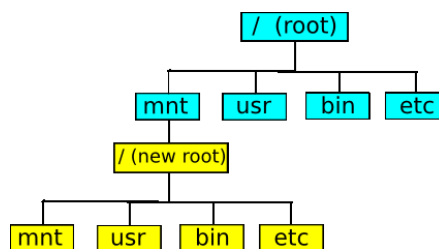


Рисунок 1 – Фрагмент ієрархії файлової системи *Ext*

Переваги використання *chroot* визначаються вимогами безпеки:

- 1) розділення привілеїв, коли *chroot* не дозволяє потенційному атакуючому нанести будь-які пошкодження ОС через програму, яку він несанкціоновано вніс у файлову систему;
- 2) створення фейкових файлових систем, коли *chroot*-каталог може бути наповнений так, щоб симулювати реальну файлову систему та запобігти витоку інформації, оскільки зломисник отримає інформацію про штучне оточення;
- 3) карантинне тестування невідомих програм, коли окрема копія ОС може бути встановлена в *chroot*-каталог як тестове оточення для програми без ризику несподіваних пошкоджень реальної ОС.

Недоліки використання *chroot*:

- 1) механізм *chroot* сам по собі не вміє здійснювати обмеження використання ресурсів (пропускна спроможність вводу-виводу, дисковий простір або час центрального процесора);

2) сам по собі механізм *chroot* не повністю безпечний, бо, якщо програма, запущена в *chroot* має привілеї *root*, вона може виконати власний виклик *chroot* для того, щоб вибратися назовні обмеження каталогу;

3) більшість *Unix*-подібних ОС не повністю орієнтовані на файлову систему і залишають потенційно руйнівну функціональність (обмін мережевими пакетами та контроль процесів), яка доступна *API* ще до програми в *chroot*.

Синтакс команди *chroot*:

```
chroot [опції] новий_root_каталог [команда]
```

chroot може виконувати окремі команди, або записати оболонку командного рядку.

Нижче наведено приклад команд, які створюють окремий каталог та розміщують в ньому програму *bash* з необхідними бібліотеками.

Крок 1. Створити каталог як майбутній *root*-каталог:

```
mkdir /tmp/blazhko_root
```

Крок 2. Створити каталог *bin* для розміщення *executable*-файлу команди *bash*, яка міститься в каталозі */bin*:

```
mkdir /tmp/blazhko_root/bin
```

```
cp /bin/bash /tmp/blazhko_root/bin
```

Крок 3. Переглянути список файлів динамічних бібліотек, на які посиляється *executable*-файл команди *bash*:

```
ldd /tmp/blazhko_root/bin/bash
```

Крок 4. Налаштувати розміщення необхідних файлів динамічних бібліотек:

```
mkdir -p /tmp/blazhko_root/lib/i386-linux-gnu/
```

```
cd /tmp/blazhko_root/lib/i386-linux-gnu/
```

```
cp /lib/i386-linux-gnu/libtinfo.so.5 ./
```

```
cp /lib/i386-linux-gnu/libdl.so.2 ./
```

```
cp /lib/i386-linux-gnu/libc.so.6 ./
```

```
cp /lib/ld-linux.so.2 ../
```

Крок 5. Змінити розташування *root*-каталогу та запустити *bash*-оболонку:

```
sudo chroot /tmp/blazhko_root /bin/bash
```

Крок 6. Виконати команди в межах нового *root*-каталогу

```
pwd
```

```
cd
```

```
echo $HOME
```

```
export HOME=/'
```

Крок 7. Повернутися до старого *root*-каталогу

exit

Можливо, що під час виконання команди *chroot* виникне наступна помилка:

chroot: failed to run command '/bin/bash': No such file or directory

Ця помилка може вказувати не на те, що файлу */bin/bash* не існує, а на те, що відсутні необхідні файли з динамічними бібліотеками, які забули скопіювати.

Після успішної зміни кореневого каталогу, можна перевірити роботу програми *bash*, виконавши команди, які є вбудованими в саму програму, наприклад, *cd*, *pwd*, *echo*, *type*, *alias*, *test*, *ulimit*, *exit*. Повний список вбудованих в *bash*-програму команд доступний за посиланням - https://www.gnu.org/software/bash/manual/html_node/Bash-Builtins.html

Але команди, які є зовнішніми по відношенню до *bash*, наприклад, команда *ls*, вже виконуватися не будуть.

1.2 Контейнерна віртуалізація

Вказані раніше недоліки механізму команди *chroot* визначили подальші напрямки у розвитку так званої *контейнерної віртуалізації*, яка надає користувачеві його власний ізольований простір (*контейнер*) для виконання однієї програми, декількох програм або увесь пакет програм ОС одного типу за принципами *chroot*-механізму:

1) *контейнерний образ (Container Image)* – ізольована файлова система, яка створюється на основі концепції роботи механізму *chroot* та містить:

- *executable*-файли, які виконують задачі користувача в ізоляції від інших задач;
- усі програмні залежності динамічних бібліотек,
- конфігурації запуску *executable*-файлів, змінні середовища;
- команди за замовчуванням для запуску.

2) *контейнер* – це екземпляр контейнерного образу, через який можна використовувати контейнерний образ.

Особливості контейнерної віртуалізації:

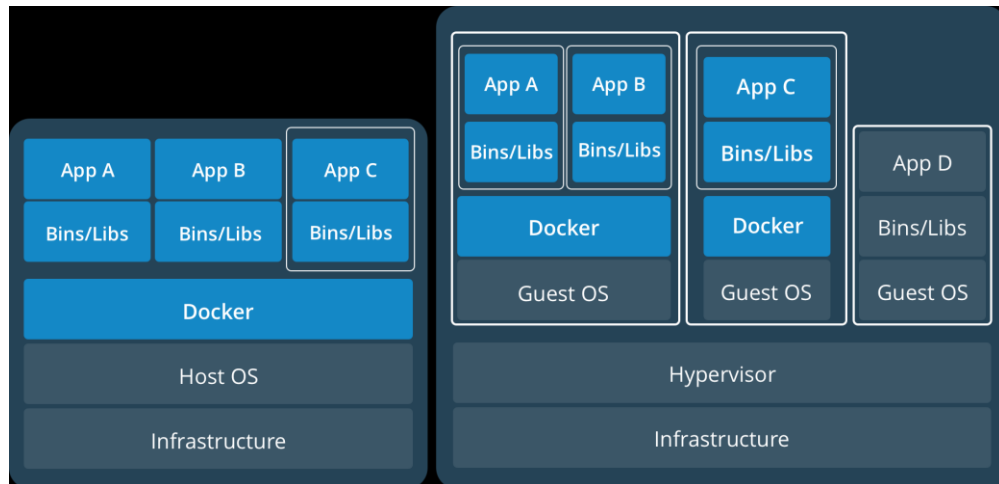
- 1) контейнер містить програмний образ (*image*) з файлами;
- 2) ядро ОС забезпечує повну ізольованість контейнерів, тому програми з різних контейнерів не можуть бачити програмні образи інших програм та впливати одна на одну;
- 3) контейнер орієнтований на роботу зі спорідненими (гомогенними) *host*-ОС та *guest*-ОС, що дозволяє підвищити швидкість роботи при відсутності додаткових процесів з емуляції віртуального обладнання при обміні повідомлень між *host*-ОС та *guest*-ОС різних типів,
- 4) можлива різноманітність типів *host*-ОС та *guest*-ОС, наприклад, в ОС *Windows* можна запустити програмне забезпечення, створене для ОС *Linux*.

1.2 Програмне забезпечення контейнерної віртуалізації *Docker*

1.2.1 Властивості контейнерної віртуалізації *Docker*

Популярним прикладом програмного забезпечення контейнерної віртуалізації є *Docker* - <https://www.docker.com/>

На рисунку 2 показано схему взаємодії процесів для контейнерної віртуалізації (a) та апаратної віртуалізації (b).



(a) – контейнерна віртуалізація

(b) – апаратна віртуалізація

Рисунок 2 – Схема взаємодії процесів для контейнерної віртуалізації (a) та апаратної віртуалізації (b)

Основні особливості *Docker*:

- можливість розміщення в ізольованому середовищі різних комбінацій виконуваних файлів, бібліотек, файлів конфігурації, скриптів тощо;
- підтримка роботи на будь-якому комп'ютері на базі архітектури *x86_64* з системою на базі ядра *Linux* поверх немодифікованих сучасних ядер *Linux* та у стандартних конфігураціях більшості популярних дистрибутивів *Linux*;
- використання легковагих контейнерів для ізоляції процесів від інших процесів;
- контейнери використовують свою власну автономну файлову систему, забезпечуючи ізоляцію процесів на рівні файлової системи;
- обмеження для кожного контейнеру рівня ізоляції ресурсів таких як витрата пам'яті, навантаження на *CPU*;
- кожен ізольований процес має доступ тільки до пов'язаного з контейнером мережевого простору імен, включаючи віртуальний мережевий інтерфейс і прив'язані до нього IP-адреси;
- можливість створення складних контейнерів через зв'язування між собою вже існуючих контейнерів.

1.2.2 Встановлення програмного забезпечення *Docker*

Для встановлення програмного пакету *Docker* на різні ОС можна розглянути вказівки:

- ОС *Windows* – <https://docs.docker.com/desktop/windows/install/>
- ОС *Linux* – <https://docs.docker.com/desktop/install/linux-install/>
- ОС *MacOS* – <https://docs.docker.com/desktop/install/mac-install/>

1.2.2.1 Особливості встановлення *Docker* в ОС *Linux*

Для проведення експериментів зі встановлення *Docker* в ОС *Linux* скористаємося репозиторієм образів різних ОС, збережених у *VDI*-форматі програми *Oracle VM VirtualBox* – <https://www.osboxes.org/virtualbox-images/>. Для всіх ОС: логін=*osboxes*, пароль=*osboxes.org*

1.2.2.1.1 *Ubuntu DeskTop 20.04.4 Focal Fossa*

Розглянемо кроки встановлення *Docker* в командному рядку на прикладі дистрибутиву *Ubuntu DeskTop 20.04.4 Focal Fossa*, встановленого через *VDI*-формат за посиланням <https://sourceforge.net/projects/osboxes/files/v/vb/55-U-u/20.04/20.04.4/64bit.7z/download>

Після встановлення дистрибутиву *Ubuntu DeskTop 20.04.4 Focal Fossa* рекомендується встановити програмний пакунок *sshd*-серверу, виконавши наступні команди:

```
sudo apt update
sudo apt install openssh-server
```

Додатково рекомендується встановити програмний пакунок *net-tools* із командою *ifconfig*, виконавши наступні команди:

```
sudo apt install net-tools
```

Для встановлення програми *Docker* необхідно виконати наступні команди менеджера програмних *apt*.

Крок 1. Оновити версії програмних пакунків за існуючим списком адрес:

```
sudo apt update
```

Крок 2. Встановити додаткові програмні пакунки:

```
sudo apt install apt-transport-https ca-certificates \
curl software-properties-common
```

Крок 3. Оновити ключі безпеки для перевірки програмних пакетів:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo apt-key add -
```

Крок 4. Додати до репозиторія програмних пакунків нову адресу, за якою розміщено програмний пакунок *Docker* для ОС *Ubuntu* версії *focal stable*:

```
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
focal stable"
```

Крок 5. Отримати перелік репозиторіїв, в яких присутній програмний пакунок:

```
apt-cache policy docker-ce
```

Крок 6. Встановити програмний пакунок

```
sudo apt install docker-ce
```

Для перевірки успішності встановлення *Docker* можна виконати запуск контейнера на основі тестового образу *hello-world*:

```
sudo docker run hello-world
```

Для встановлення програми *Docker Desktop* необхідно виконати команди:

```
curl -O https://desktop.docker.com/linux/main/amd64/docker-desktop-4.20.1-amd64.deb
```

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

```
sudo apt install ./docker-desktop-4.20.1-amd64.deb
```

Програму можна зупустити через пошукову графічну консоль або з командного рядку:

```
systemctl --user start docker-desktop
```

Для роботи з *Docker* за замовчуванням необхідні права адміністратора ОС. Але в ОС *Linux* можна дозволити роботу звичайному користувачу через додавання його облікового запису до групи *docker*, наприклад

```
usermod -a -G docker user1
```

1.2.2.1.2 CentOS 7.9

Розглянемо кроки встановлення *Docker* в командному рядку на прикладі дистрибутиву *CentOS версії 7.9*, встановленого через *VDI*-формат за посиланням

https://sourceforge.net/projects/osboxes/files/v/vb/10-C-nt/7/7.9-2009/CentOS-7.9-2009_VB-64bit.7z/download

Для встановлення програми *Docker* необхідно виконати команди менеджера програмних *RPM*-пакунків *Yellowdog Updater, Modified (YUM)*:

```
sudo yum install -y yum-utils
```

```
sudo yum-config-manager \
```

```
--add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

```
sudo yum install docker-ce docker-ce-cli containerd.io
```

```
sudo systemctl start docker
```

Для перевірки успішності встановлення *Docker* можна виконати запуск контейнера на основі тестового образу:

```
sudo docker run hello-world
```

1.2.2.1.3 *Fedora 33 Workstation*

Розглянемо кроки встановлення *Docker* в командному рядку на прикладі дистрибутиву *Fedora 33 Workstation*, встановленого через *VDI*-формат за посиланням

https://sourceforge.net/projects/osboxes/files/v/vb/18-F-d/33/Fedora-33-Workstation-VB_64bit.7z/download

Після встановлення дистрибутиву *Fedora 33 Workstation* рекомендується увімкнути *sshd*-даїмон, використовуючи наступні команди:

```
sudo systemctl enable sshd
sudo systemctl start sshd
```

Для встановлення програми *Docker* необхідно виконати команди менеджера програмних пакунків *Dandified Yum (DNF)*, який розроблено у проєкті *Fedora* та впроваджено у дистрибутив *Red Hat Enterprise Linux версії 8* і *CentOS версії 8* на заміну менеджера *Yum*:

```
sudo dnf -y install dnf-plugins-core
sudo dnf config-manager \
--add-repo https://download.docker.com/linux/fedora/docker-ce.repo
sudo dnf install docker-ce docker-ce-cli containerd.io
sudo systemctl start docker
```

Для перевірки успішності встановлення *Docker* можна виконати запуск контейнера на основі тестового образу:

```
sudo docker run hello-world
```

Для встановлення програми *Docker Desktop* необхідно виконати команди:

```
curl -O https://desktop.docker.com/linux/main/amd64/docker-desktop-4.20.1-x86_64.rpm
sudo dnf install ./docker-desktop-4.20.1-x86_64.rpm
systemctl --user start docker-desktop
```

1.2.2.2 Особливості встановлення *Docker* в ОС *Windows*

Напочатку свого існування програмне забезпечення *Docker* використовувалося для створення контейнерної віртуалізації під керуванням *Linux host*-ОС для виконання контейнерів в *Linux guest*-ОС, тобто воно забезпечувало гомогенну контейнерну віртуалізацію.

Але для 64-бітної ОС *Windows 10* було створено *Windows Subsystem for Linux (WSL)* як програмний прошарок для виконання *executable*-файлів ОС *Linux* в ОС *Windows 10*. В другій версії *WSL (WSL2)* замість прошарку трансляції системних програмних викликів вже використовується справжнє ядро ОС *Linux*, яке працює разом з гіпервізором *Hyper-V*. Тому рекомендується пересвідчитися, що *Hyper-V* увімкнено, про що можна дізнатися на сторінці:

<https://docs.docker.com/desktop/troubleshoot/topics/#virtualization>

В урахуванням впровадження *WSL2* тепер програмне забезпечення *Docker* вже забезпечує гетерогенну контейнерну віртуалізацію, коли *host-OC* за архітектурою може відрізнятися від *guest-OC*.

Перед встановленням *Docker* необхідно оновити *WSL*, виконавши наступні кроки.

Крок 1. Запустити програму *PowerShell* від імені адміністратора (*Start menu > PowerShell > right-click > Run as Administrator*)

Крок 2. Включити опцію використання "*Windows Subsystem for Linux*":

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

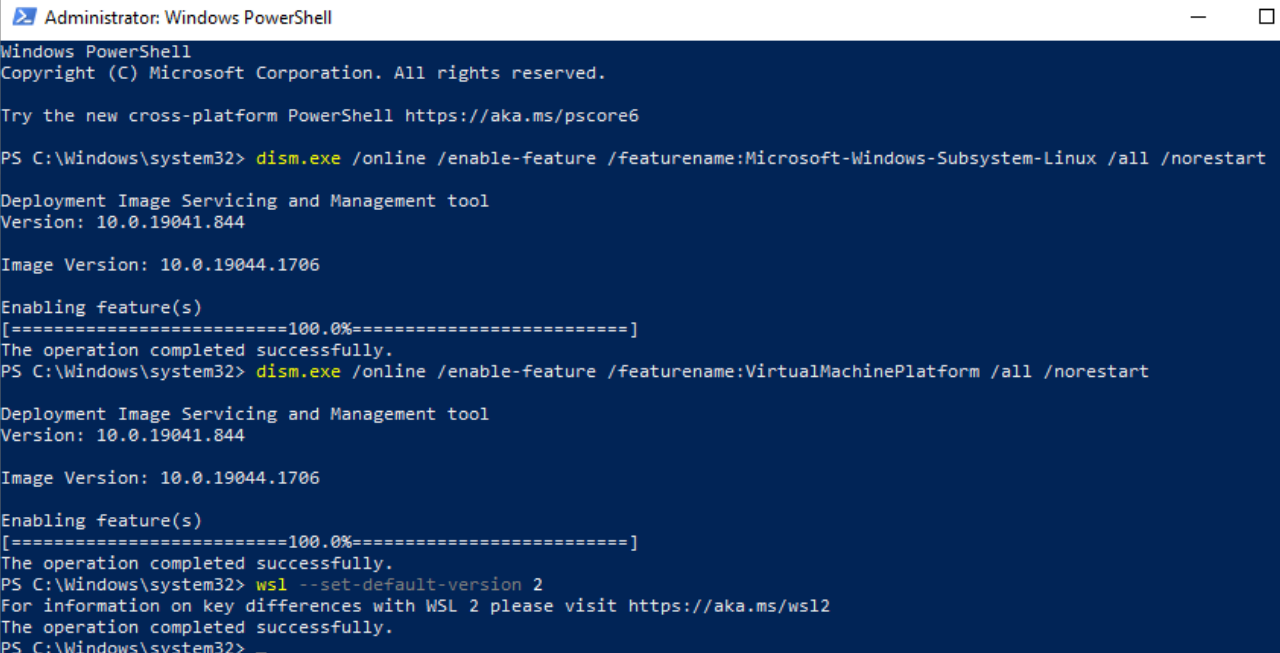
Крок 3. Включити опцію використання *Virtual Machine Platform*:

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

Крок 4. Включення другої версії *WSL*:

```
wsl --set-default-version 2
```

На рисунку 3 показано фрагмент екрану зі вказаними вище кроками



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

Deployment Image Servicing and Management tool
Version: 10.0.19041.844

Image Version: 10.0.19044.1706

Enabling feature(s)
[=====100.0%=====]
The operation completed successfully.
PS C:\Windows\system32> dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

Deployment Image Servicing and Management tool
Version: 10.0.19041.844

Image Version: 10.0.19044.1706

Enabling feature(s)
[=====100.0%=====]
The operation completed successfully.
PS C:\Windows\system32> wsl --set-default-version 2
For information on key differences with WSL 2 please visit https://aka.ms/ws12
The operation completed successfully.
PS C:\Windows\system32>
```

Рис. 3 – Фрагмент екрану із кроками встановлення *WSL*

Після успішності виконання вказаних кроків можна виконати інсталяцію *Docker*, інсталяційний пакет якої отримано за посиланням -

<https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe>

Для перевірки успішності встановлення *Docker* в командному рядку можна виконати запуск контейнера на основі тестового образу *hello-world*:

```
docker run hello-world
```

1.2.3 Керування контейнерною віртуалізацією через графічний інтерфейс

Docker дозволяє керувати контейнерами через:

- графічну інтерфейс– програма *Docker Desktop*;
- інтерфейс командного рядку – утиліта *docker*.

На рисунку 4 показано фрагмент програми *Docker*, запущеною в ОС *Windows*.

На екрані показано декілька популярних образів з для ОС *Linux*. Для встановлення образу в локальний репозиторій необхідно натиснути кнопку «Run».

Перелік встановлених образів можна переглянути на вкладці «*Images*».

Перелік створених контейнерів образів можна переглянути на вкладці «*Containers*».

Якщо створений контейнер використовує зовнішні дані, тоді інформація про це вказується на вкладці «*Volumes*».

Додатково *Docker*-образи можна знайти у репозиторії за окремим посиланням <https://hub.docker.com/>

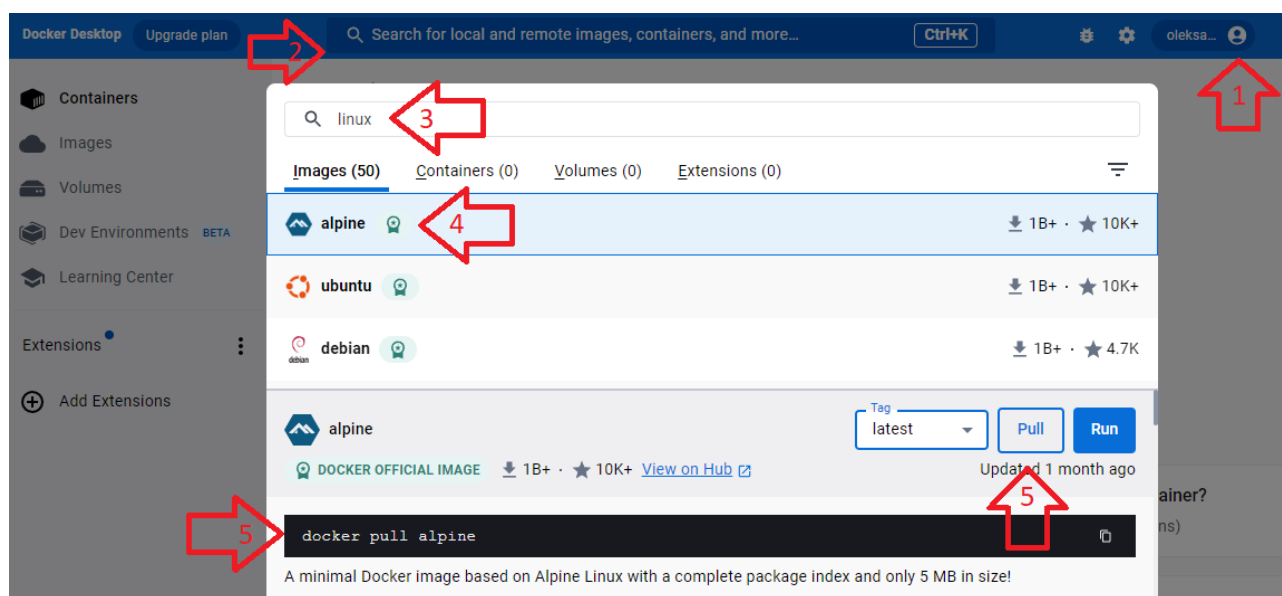


Рис. 4 – Фрагмент екрану вибору *Docker*-образу для ОС *Linux*

На рисунку 5 показано процес вибору та завантаження *Docker*-образу для СКБД *PostgreSQL* через кнопку *Pull*.

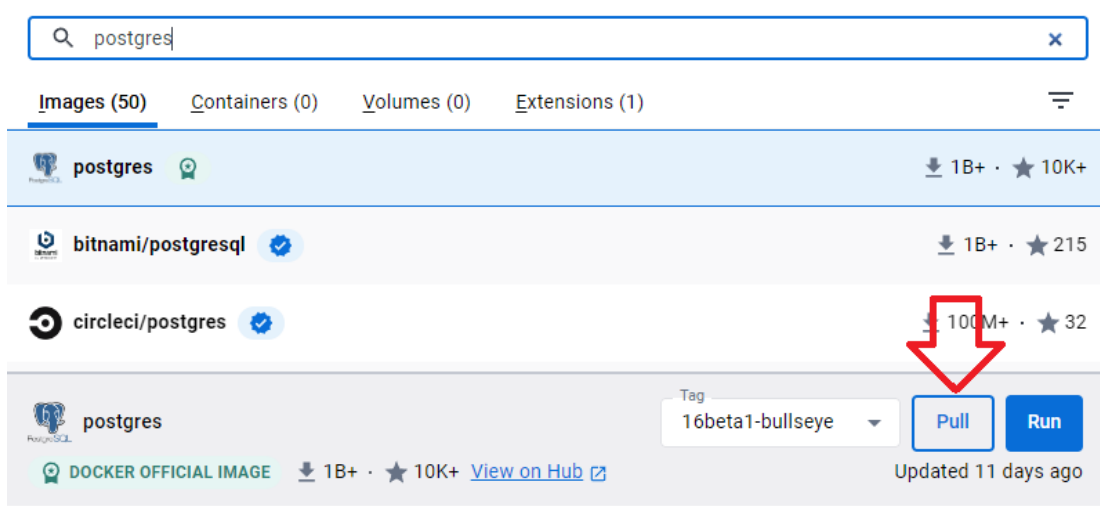


Рис. 5 – Фрагмент екрану вибору *Docker*-образу для СКБД *PostgreSQL*

На рисунку 6 показано фрагмент екрану зі вже завантаженим *Docker*-образом для СКБД *PostgreSQL*, для якого можна створити та запустити контейнер.

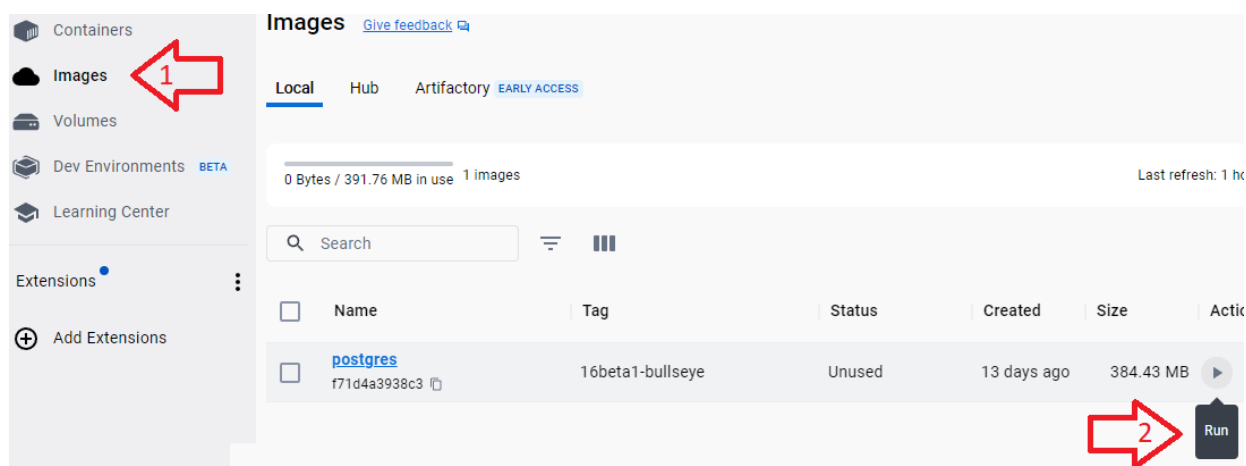


Рис. 6 – Фрагмент екрану початку процесу створення контейнеру

На рисунку 7 наведено фрагмент екрану визначення опцій контейнеру перед стартом контейнеру через кнопку *Run*.

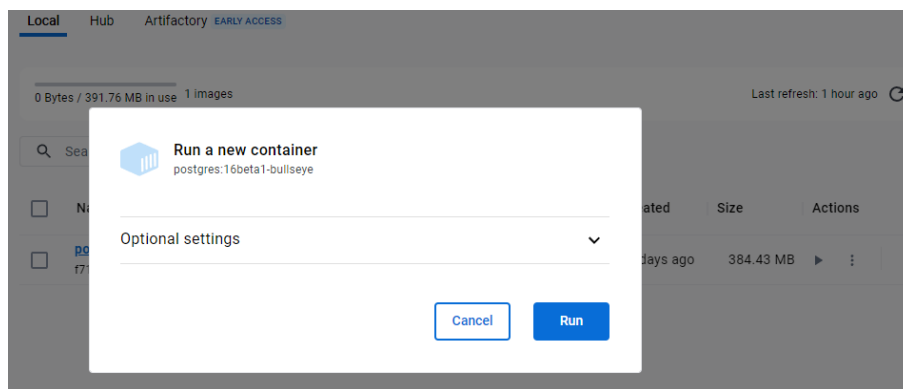


Рис. 7 – Фрагмент екрану визначення опцій контейнеру

Якщо не внести значення опцій запуску контейнеру та зразу натиснути кнопку *Run*, тоді буде створено контейнер з опціями за замовчуванням із наступним його запуском. На рисунку 8 наведено рядки із помилкою запущеного контейнеру, для якого автоматично створено назву.

Але такий швидкий та помилковий варіант створення контейнеру має свої переваги через огляд опцій запуску контейнеру, про які недосвідчений користувач може не знати. Для перегляду опцій запуску контейнеру необхідно перейти до вкладинки *Inspect*.

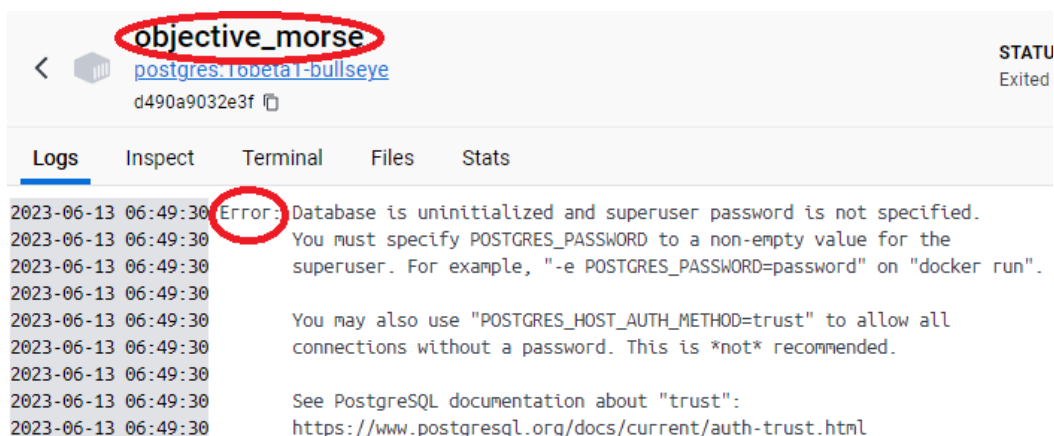


Рис. 8 – Фрагмент екрану з описом контейнеру у вкладинці *Logs*

На рисунку 9 наведено фрагмент екрану із значеннями опцій запуску контейнеру, виставлених за замовчуванням.

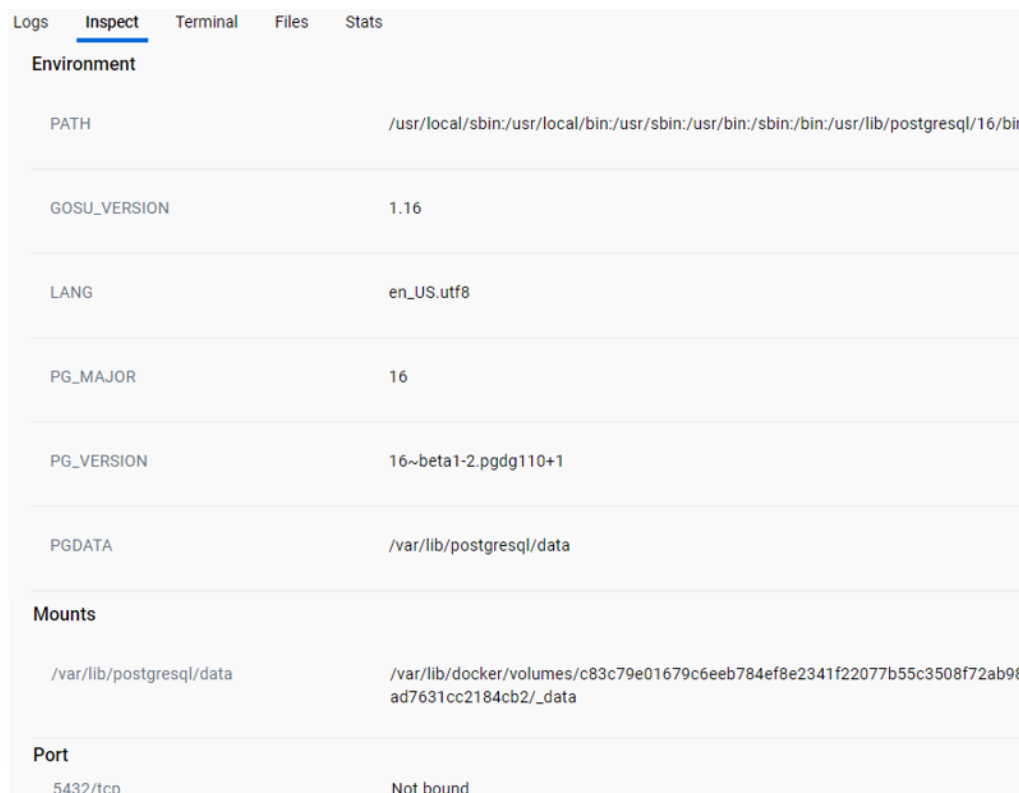


Рис. 9 – Фрагмент екрану із опціями запуску контейнеру за замовчуванням

Для створеного контейнеру не можна змінювати значення параметрів запуску. Тому треба його видалити та повторно запустити. На рисунку 10 наведено приклад визначення опцій контейнеру:

- назва контейнеру;
- опція `POSTGRES_PASSWORD=1234` зі значенням змінної середовища запуску серверу СКБД *PostgreSQL* для виправлення помилки «*superuser password is not specified*».

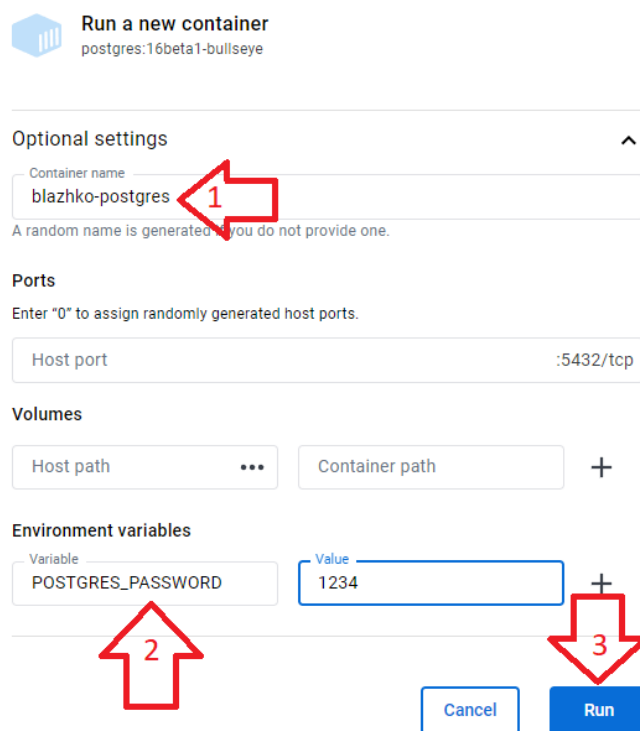


Рис. 10 – Фрагмент екрану із прикладом визначення опцій контейнеру

Після успішного запуску контейнера з'являться повідомлення успішно запущеного сервера СКБД *PostgreSQL*, фрагмент яких показано на рисунку 11.

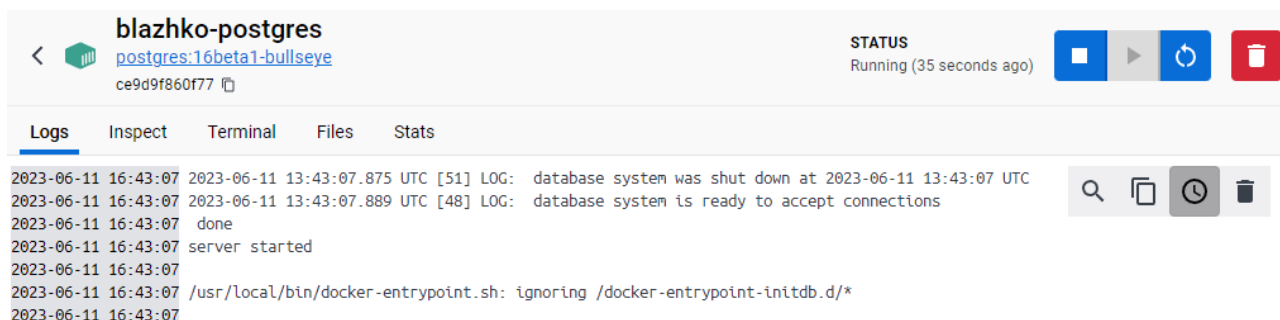


Рис. 11 – Фрагмент екрану із успішно запущеним контейнером

Для виконання команд у запущеному контейнері можна перейти до вкладки *Terminal*, як показано на рисунку 12.

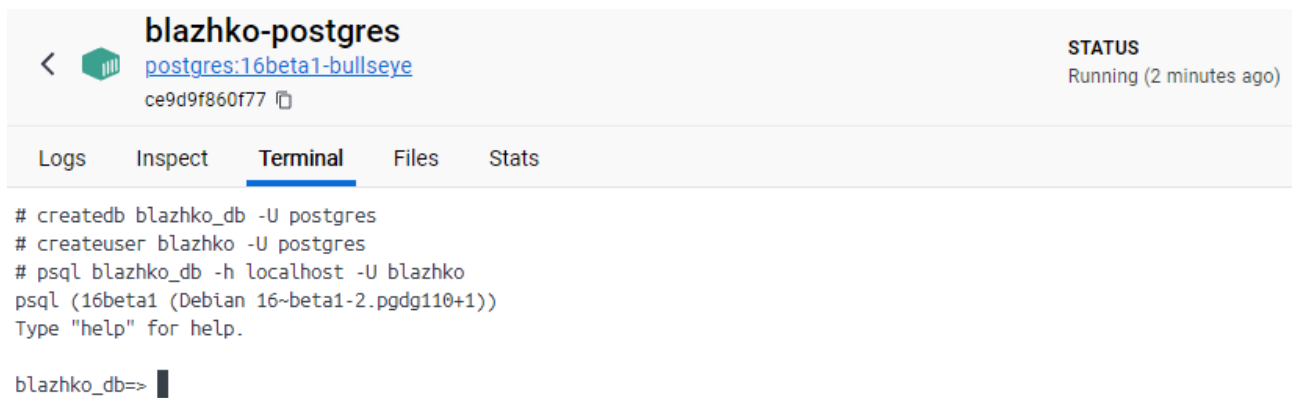


Рис. 12 – Фрагмент екрану із виконанням команд у запущеному контейнері

1.2.4 Керування контейнерною віртуалізацією через інтерфейс командного рядку

Наведемо приклади основних команд роботи з *Docker*-контейнерами.

Для пошуку контейнерного образу використовується команда:

```
docker search рядок
```

Більшість програм мають версії або релізи, тому їх *Docker*-образи також можна отримати за версією, яка в термінології *Docker* називається *tag*, який вказується після назви через двокрапку. За замовчуванням, якщо значення *tags* не вказувати, тоді буде отримано останню стабільну версію, яка має *tag=latest*.

Для отримання переліку версій контейнерного образу програми можна виконати наступний конвейер команд, де перша команда отримує перелік записів у *json*-форматі, друга команда розміщує описи в окремих рядках, а третя команда вже вибирає рядки з описом тегів:

```
curl https://registry.hub.docker.com/v2/repositories/library/\
<програма>/tags?page_size=10000 \
| tr ',' '\n' \
| grep -E '"name"'
```

Наприклад, для завантаження переліку версій контейнерного образу програми *ubuntu* необхідно виконати наступний конвейер команд:

```
curl https://registry.hub.docker.com/v2/repositories/library/\
ubuntu/tags?page_size=10000 \
| tr ',' '\n' \
| grep -E '"name"'
```

Для завантаження з репозиторію контейнерного образу використовується команда:
docker pull образ

Наприклад, для завантаження образу контейнера з ОС *Ubuntu* останньої стабільної версії, необхідно виконати:

```
docker pull ubuntu:latest
```

Наприклад, для завантаження образу контейнера з ОС *Ubuntu* версії 14.04, необхідно виконати:

```
docker pull ubuntu:14.04
```

Для запуску контейнера на основі контейнерного образу використовується команда:

```
docker run [опції] образ команда
```

Приклади опцій команди *run*:

- *--name* вказує на назву контейнера;
- *-i* (скорочення для *--interactive*) – забезпечення інтерактивний режим роботи з використанням потоку *STDIN*;
- *-t* (скорочення для *--tty*) – створення псевдотерміналу, який з'єднає термінал, що використовується, з потоками *STDIN* та *STDOUT* контейнера;
- *-d* (скорочення для *--detach*) – запуск контейнера у фоновому режимі;
- *--e* (скорочення для *--env*) – встановлення значень змін середовища;
- *--env-file [path-to-env-file]* – встановлення значень змін середовища з файлу;
- *--ulimit* встановлення опцій команди *Ulimit* з обмеження ресурсів;
- *--rm* автоматичне видалення контейнеру після завершення роботи з ним.

Якщо контейнер запускається в ОС *Windows* через оболонку *Git-Bash*, тоді у разі використання опцій *-it* можлива поява помилки: *the input device is not a TTY. If you are using mintty, try prefixing the command with 'winpty'*. В цьому випадку перед командою *docker* треба вказати команду *winpty*.

Наприклад, для запуску контейнера з назвою *ubuntu-for-blazhko*, який використовує останню стабільну версію контейнерного образу *ubuntu* для ОС *Ubuntu* для подальшої взаємодії з користувачем (опції *-it*) після чого в контейнері буде запущено команду *bash*, необхідно виконати:

```
docker run \  
--name ubuntu-for-blazhko \  
-it \  
ubuntu:latest bash
```

Після отримання доступу до терміналу можна виконати команду перегляду опису дистрибутиву ОС:

```
cat /etc/os-release
```

Для завершення роботи зі створеним контейнером у псевдотерміналі можна виконати команду *exit*.

Для отримання списку запущених контейнерів використовується команда:

```
docker ps [-a]
```

Опція *-a* дозволяє отримати список створених, але не запущених контейнерів.

Для призупинення роботи контейнера за назвою або ідентифікатором використовується команда:

```
docker stop контейнер
```

Для повторного запуску призупиненого контейнера виконується команда:

```
docker start контейнер
```

Для подальшого виконання команд у контейнері використовується команда:

```
docker exec [опції] контейнер команда
```

Прикладами опцій команди *exec* можуть бути опції команди *run*: *-t*, *-d*, *--e*.

Наприклад, для виконання команди */bin/bash* у запущеному контейнері з назвою *ubuntu-for-blazhko* та подальшої взаємодії з користувачем (опції *-it*), необхідно виконати:

```
docker exec \  
-it \  
ubuntu-for-blazhko \  
bash
```

Для видалення створеного але не запущеного контейнеру (за назвою або ідентифікатором) використовується команда:

```
docker rm контейнер
```

1.2.5 Приклад роботи з СКБД *PostgreSQL* через *Docker*-контейнер

Наведемо приклад роботи з СКБД *PostgreSQL* через *Docker*-контейнер.

Контейнер можна знайти у репозиторії <https://hub.docker.com/>, наприклад, за адресою https://hub.docker.com/_/postgres

Виконання завдання передбачає наступні кроки.

Крок 1. Завантажити образ контейнера *postgres* останньої стабільної версії через команду:

```
docker pull postgres
```

Крок 2. Запустити образ *postgres* зі створенням контейнеру за назвою *blazhko-postgres*, паролем адміністратора (опція *-e POSTGRES_PASSWORD=1234*) та подальшою роботою сервера СКБД *PostgreSQL* у фоновому режимі (опція *-d postgres*):

```
docker run \  
--name blazhko-postgres \  
-e POSTGRES_PASSWORD=1234 \  
-d postgres \  
postgres
```

Крок 3. Виконати в контейнері *blazhko-postgres* команду *bash* для початку роботи з оболонкою командного рядку:

```
docker exec -it blazhko-postgres bash
```

Крок 4. В оболонці командного рядку виконати команду *createdb* зі створення БД *blazhko_db* від імені адміністратора *postgres*:

```
createdb blazhko_db -U postgres
```

Крок 5. В оболонці командного рядку виконати команду *createuser* зі створення користувача *blazhko* СКБД *PostgreSQL*

```
createuser blazhko -U postgres
```

Крок 6. В оболонці командного рядку виконати команду *psql* для встановлення з'єднання з БД *blazhko_db* від імені *blazhko*:

```
psql blazhko_db -h localhost -U blazhko
```

Крок 7. Завершити роботу з командою *psql* через команду *\q* та завершити роботу з контейнером через команду *exit*

Крок 8. Зупинити роботу контейнера *blazhko-postgres*:

```
docker stop blazhko-postgres
```

Крок 9. Видалити контейнер *blazhko-postgres*:

```
docker rm blazhko-postgres
```


2 Завдання до лабораторної роботи

Як в лабораторній роботі №14, у цій лабораторній роботі завдання виконуються на вашому локальному комп'ютері.

Виконати наступні дії з підготовки до виконання завдань роботи:

- 1) використовуючи команди *Bash*, перейти до каталогу *Git*-репозиторія, який використовувався у лабораторній роботі №14;
- 2) за необхідністю оновити зміст *Git*-репозиторія змістом з *GitHub*-репозиторію командою *pull*;
- 3) створити нову *Git*-гілку з назвою «*Laboratory-work-15*»;
- 4) перейти до роботи зі створеною гілкою;
- 5) створити каталог з назвою «*Laboratory-work-15*»;
- 6) перейти до каталогу «*Laboratory-work-15*»;
- 7) в каталозі «*Laboratory-work-15*» створити файл *README.md* та додати до файлу рядок тексту із темою лабораторної роботи «*Основи контейнерної віртуалізації програмного забезпечення*» як заголовок 2-го рівня *Markdown*-форматування;
- 8) виконати операції з оновлення *GitHub*-репозиторію змінами *Git*-репозиторія через послідовність *Git*-команд *add*, *commit* із коментарем «*Changed by Local Git*» та *push*;
- 9) на веб-сервісі *GitHub* перейти до створеної гілки «*Laboratory-work-15*»;
- 10) перейти до каталогу «*Laboratory-work-15*» та розпочати процес редагування файлу *README.md*
- 11) в подальшому за результатами рішень кожного наступного розділу завдань до файлу *README.md* додавати:
 - рядки як заголовки 3-го рівня *Markdown*-форматування з назвами розділу;
 - знімки екранів, які демонструють успішність виконання пункту завдання;
 - підписи під кожним знімком екрану із повним описом пункту завдання;

2.1 Встановлення програми контейнерної віртуалізації Docker

2.1.1 На вашому локальному комп'ютері встановити програмне забезпечення *Docker* під будь-яку *host*-ОС (*Windows*, *Linux*, *MacOS*).

Примітка: за бажанням ви можете також використати віртуальну машину *Oracle VM VirtualBox*, встановивши більш сучасну ОС *Linux* через існуючий *VDI*-образ, отриманий за адресою <https://www.osboxes.org/virtualbox-images/>, наприклад *Ubuntu 20.04.4 Focal Fossa*.

2.1.2 Перевірити успішність встановлення *Docker* через запуск контейнера на основі тестового образу *hello-world*.

2.2 Керування контейнерною віртуалізацією ОС *Linux* через інтерфейс командного рядку

2.2.1 Через командний рядок виконати пошук контейнерного образу ОС *Linux*.

2.2.2 Отримати перелік версій контейнерного образу ОС *fedora*, назва яких містить тільки цифри.

2.2.3 Завантажити контейнерний образ ОС *fedora* версії, номер якої відповідає останнім цифрам вашої групи, наприклад, 21, 22, 23.

2.2.4 Запустити контейнер з образом, завантаженим у попередньому пункті, який має назву «*fedora-for-surname*», де *surname* – ваше прізвище транслітерацією, наприклад «*fedora-for-blazhko*», для подальшої взаємодії з користувачем через команду *bash*.

2.2.5 Після отримання доступу до терміналу ОС переглянути опис дистрибутиву ОС.

2.2.6 В другому псевдотерміналі *host*-ОС виконати команду отримання списку запущених контейнерів.

2.2.7 В першому псевдотерміналі завершити роботу із контейнером.

2.2.8 В другому псевдотерміналі *host*-ОС виконати команду отримання списку створених контейнерів.

2.2.9 Видалити створений контейнер.

2.3 Керування файловою віртуалізацією ОС *Linux* через команду *chroot*

Повторити пункт 2.2.4 з попереднього розділу завдань.

2.3.1 Створити каталог за шаблоном */tmp/surname_root*, де *surname* – ваше прізвище транслітерацією, наприклад, */tmp/blazhko_root*

2.3.2 Розмістити у створеному каталозі програму *bash* та всі необхідні файли динамічних бібліотек.

2.3.3 Змінити розташування *root*-каталогу на створений каталог.

2.3.4 Спробувати змінити каталог на каталог поточного користувача.

2.3.5 Отримати назву поточного каталогу.

2.3.6 Змінити значення змінної *HOME* на назву поточного каталогу.

2.3.7 Змінити каталог на каталог поточного користувача.

2.3.8 Спробувати виконати команду отримання переліку файлів каталогу.

2.3.9 Повернутися до старого *root*-каталогу.

2.4 Керування контейнерною віртуалізацією СКБД *PostgreSQL* через інтерфейс командного рядку

2.4.1 Завантажити образ контейнера *postgres* останньої стабільної версії.

2.4.2. Запустити образ *postgres* зі створенням контейнеру з назвою за шаблоном «*surname-postgres*», де *surname* – ваше прізвище транслітерацією, паролем адміністратора 5432 та подальшою роботою сервера СКБД *PostgreSQL* у фоновому режимі

2.4.3 Виконати в контейнері *surname-postgres* команду *bash*, щоб розпочати роботу із оболонкою командного рядку.

2.4.4 В оболонці командного рядку створити БД із назвою за шаблоном «*surname_db*»

2.4.5 В оболонці командного рядку створити користувача СКБД *PostgreSQL* за шаблоном «*surname*».

2.4.6 В оболонці командного рядку встановити з'єднання зі створеною БД від імені створеного користувача.

2.4.7 Завершити роботу з командою *psql* та завершити роботу з контейнером.

2.4.8 Виконати команду отримання списку запущених контейнерів.

2.4.9 Зупинити роботу контейнера *surname-postgres*.

2.4.10 Видалити контейнер *surname-postgres*.

2.5 Підготовка процесу *Code Review* для надання рішень завдань лабораторної роботи на перевірку викладачем

На веб-сервісі *GitHub* зафіксувати зміни у файлі *README.md*.

Скопіювати файли, які було створено у попередніх розділах завдань, в каталог «*Laboratory-work-15*» *Git*-репозиторію.

Оновити *Git*-репозиторій змінами нової гілки «*Laboratory-work-15*» з *GitHub*-репозиторію.

Оновити *GitHub*-репозиторій змінами нової гілки «*Laboratory-work-15*» *Git*-репозиторію. Виконати запит *Pull Request*.

Примітка: Увага! Не натискайте кнопку «Merge pull request»!

Це повинен зробити лише рецензент, який є вашим викладачем!

Рецензент-викладач перегляне рішення та надасть оцінку та закрий *Pull Request* з операцією *Merge*.

2.6 Оцінка результатів виконання завдань лабораторної роботи

Оцінка	Умови
+2 бали	під час <i>Online</i> -заняття виконано правильні рішення завдань
+6 бали	1) всі рішення роботи відповідають завданням 2) <i>Pull Request</i> представлено не пізніше години після завершення <i>Online</i> -заняття
-0.5 балів за кожну помилку	в рішенні є помилка, про яку вказано в <i>Code Review</i>
-1 бал	<i>Pull Request</i> представлено пізніше ніж через годину після завершення <i>Online</i> -заняття

Література

1. Блажко О.А. Відео-запис лекції «Основи контейнерної віртуалізації програмного забезпечення». URL: https://youtu.be/_y-4bYIU-NI