ЗАТВЕРДЖУЮ
Начальник кафедри
інформаційних технологій та систем
електронних комунікацій
к.т.н., доцент
Олександр ПРИДАТКО
«____»_____20___ р.

МЕТОДИЧНА РОЗРОБКА
для проведення лабораторного заняття з дисципліни
«Клієнт-серверне програмування»

Лабораторна робота 7

Встановлення Docker контейнера на базі віртуальної машини.

Мета роботи: набути навичок встановлювати додаток Docker в ОС Linux, ознайомленні з репозиторієм Docker Hub, надати основи для роботи з образами та контейнерами, написання скрипта Dockerfile для створення контейнеру.

1. Теоретичні відомості

Docker – це ПЗ з відкритим вихідним кодом, яке застосовується для розробки, тестування, доставки і запуску веб-додатків в середовищах з підтримкою контейнеризації. Він потрібен для більш ефективного використання системи і ресурсів, швидкого розгортання готових програмних продуктів, а також для їх масштабування і переносу в інші середовища з гарантійним збереженням стабільної роботи.

Основний принцип роботи Docker – контейнеризація. Цей тип віртуалізації дозволяє упаковувати програмне забезпечення по ізольованим середовищам - контейнерам. Кожен з цих віртуальних блоків отримає всі необхідні елементи для роботи програми. Це дає можливість одночасного запуску великої кількості контейнерів на одному хості.

Контейнер створюється і складається з образів. Його можна легко видалити і знову створити за короткий проміжок часу.

Образ – базовий елемент кожного контейнера.

Образи зберігаються у реєстрі. Реєстр – це сервер, на якому зберігаються образи.

Образи контейнерів являють собою набір файлів, організованих у стек шарів, розміщених на локальному комп’ютері або у віддаленому реєстрі контейнерів. Образ контейнера складається з:

– файлів ОС режиму користувача, необхідних для підтримання додатка;

– будь-яких середовищ виконання або залежностей додатків;

– будь-якого іншого файлу конфігурації, потрібного для правильної роботи додатка.

Платформа Docker складається із двох окремих компонентів:

• Docker Engine - механізму, що відповідає за створення і функціонування контейнерів, це клієнт-серверний додаток;

• Docker Hub хмарного сервісу для поширення контейнерів.

Механізм Docker Engine надає ефективний і зручний інтерфейс для запуску контейнерів.

Docker Hub – публічний репозиторій з інтерфейсом, який надається Docker Inc (назва компанії розробника Docker Hub). Цей ресурс є джерелом «офіційних» образів, розроблених командою Докер або створених у співпраці з розробником ПЗ.

Основними складовими частинами архітектури Docker є:

– сервер, який містить сервіс Docker, образи та контейнери; сервіс зв’язується з Registry, образи – метадані додатків, які запускаються в контейнерах Docker;

– контейнери – ізольовані за допомогою технологій ОС користувацькі оточення, в яких виконуються додатки. Контейнер Docker запускається з образу додатка. Розробники Docker дотримуються єдиного принципу: один контейнер – це один додаток;

– образи (Images) – шаблони додатків, які доступні тільки для читання. Поверх існуючих образів можуть додаватися нові рівні образів, які спільно представляють файлову систему, змінюючи або доповнюючи попередній рівень. Зазвичай новий образ створюється або за допомогою зберігання вже запущеного контейнера в новий образ, або за допомогою спеціальних інструкцій для утиліти Dockerfile.

– REST API, що визначає інтерфейси, які програми можуть використовувати для спілкування з демоном і вказувати йому що робити;

– клієнт інтерфейсу командного рядка (CLI) (команда docker). Застосовується для запуску різних дій на сервері Docker;

– інтерфейс командного рядка використовує API-інтерфейс Docker REST для керування або взаємодії з додатками-демонами Docker з використанням базових API і CLI;

– реєстри (registry), використовуються для зберігання образів й містять репозиторії (repository) образів, тобто це мережеві сховища образів. Вони можуть бути як приватними, так і загальнодоступними.

Dockerfile - скрипт, який дозволяє автоматизувати процес побудови контейнерів шляхом виконання відповідних команд (дій) в base образі для формування нового образу.

Усі подібні файли починаються з позначення **FROM** так як і процес побудови нового контейнера, далі йдуть різні методи, команди, аргументи або умови, після застосування яких створиться Docker контейнер.

2. Приклад виконання роботи

1. Запустити віртуальну машину з OS Linux та PuTTY
2. Вимкнути брандмауер (firewalld)

 # service firewalld stop
systemctl stop firewalld.service
 # disable firewalld
systemctl disable firewalld
 # see status
systemctl status firewalld

```
[root@ksp ~]# systemctl stop firewalld.service
[root@ksp ~]# systemctl disable firewalld
Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
[root@ksp ~]# systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; disabled; vendor p
reset: enabled)
   Active: inactive (dead)
     Docs: man:firewalld(1)
```

 # disable SELinux
setenforce 0
 # У файлі біля SELINUX поставити disabled і не потрібно кожного разу писати setenforce 0 SELINUX=disabled
nano /etc/sysconfig/selinux
 # змінити SELINUX=enforsing
 # на SELINUX=disabled

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
```

3. Оновити репозиторій epel

yum -y install epel-release
 # для оновлення необхідно зайти у файл
nano /etc/resolv.conf
та прописати DNS
nameserver 8.8.8.8

```
# Generated by NetworkManager
search localdomain
nameserver 8.8.8.8
nameserver 192.168.65.2
```

```
Running transaction
  Updating   : epel-release-7-14.noarch                                    1/2
  Cleanup    : epel-release-7-11.noarch                                    2/2
  Verifying  : epel-release-7-14.noarch                                    1/2
  Verifying  : epel-release-7-11.noarch                                    2/2

Updated:
  epel-release.noarch 0:7-14

Complete!
```

4. Встановити Docker

 # Встановити docker з сайту docker.com

curl -fsSL https://get.docker.com/ | sh

-f, --fail (Fail silently - Тихий збій) (без виведення) у разі помилок HTTP

-S, --show-error (Show error - Показувати помилку), навіть якщо використовується -s

-L, --location Слідкуйте за переспрямуваннями

-s, --silent (Silent mode -Тихий режим)

```
[root@ksp ~]# curl -fsSL https://get.docker.com/ | sh
# Executing docker install script, commit: a8a6b338bdfedd7ddefb96fe3e7fe7d40
36d945a
+ sh -c 'yum install -y -q yum-utils'
```

 # при виникненні проблем з DNS необхідно зайти у файл

nano /etc/resolv.conf

та прописати DNS

nameserver 8.8.8.8

nameserver 8.8.4.4

```
# Generated by NetworkManager
search localdomain
nameserver 8.8.8.8
nameserver 8.8.4.4
nameserver 192.168.65.2
```

 # встановити unzip, та мережеві інструменти net-tools

yum install -y unzip

yum install -y net-tools

 # запускаємо docker

systemctl start docker

 # додаємо до автозавантаження

systemctl enable docker

```
[root@ksp ~]# systemctl start docker
[root@ksp ~]# systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.serv
ice to /usr/lib/systemd/system/docker.service.
[root@ksp ~]#
```

 # перевіряємо статус

systemctl status docker

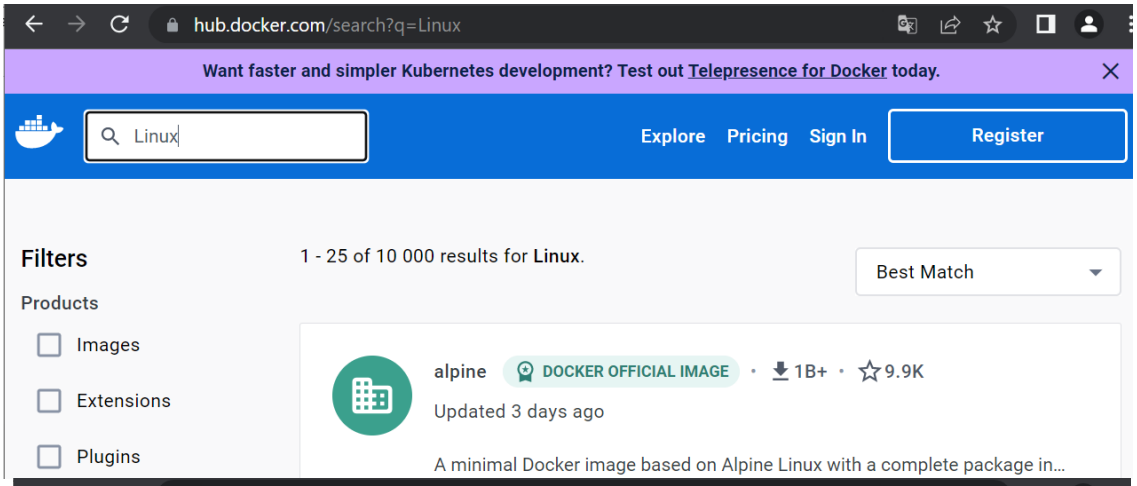
```
[root@ksp ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor p
reset: disabled)
   Active: active (running) since Sat 2023-05-13 11:04:14 EEST; 2min 14s ago
     Docs: https://docs.docker.com
 Main PID: 2044 (dockerd)
```

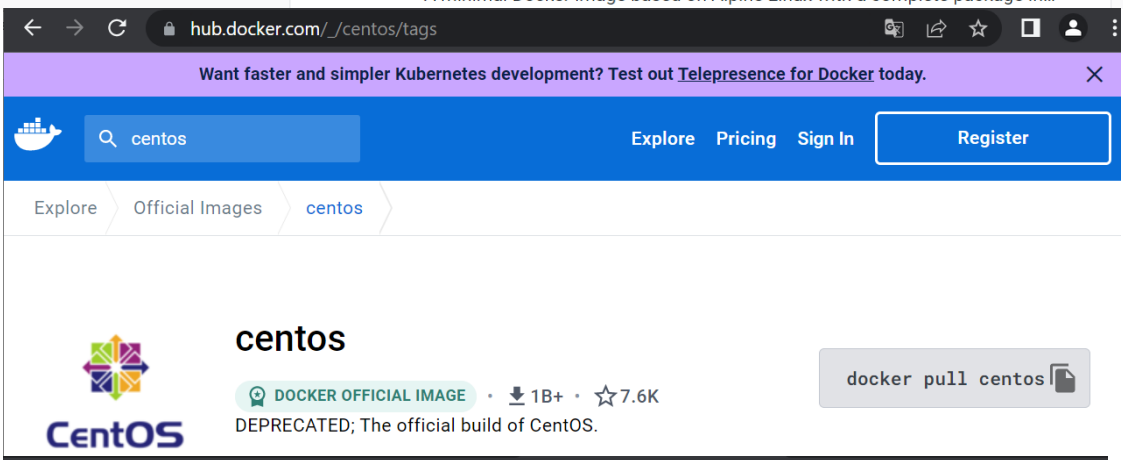
 # перевіряємо наявність образів images

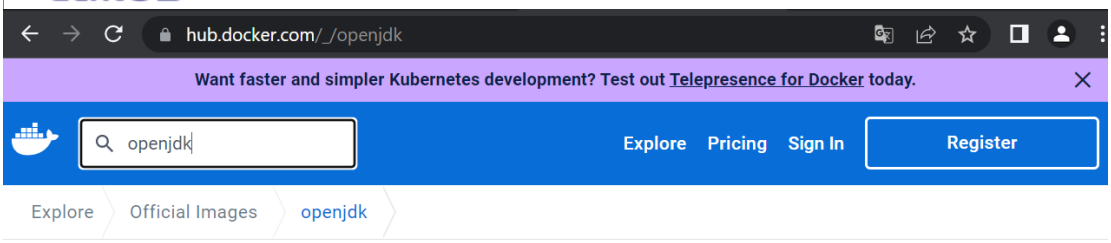
docker images

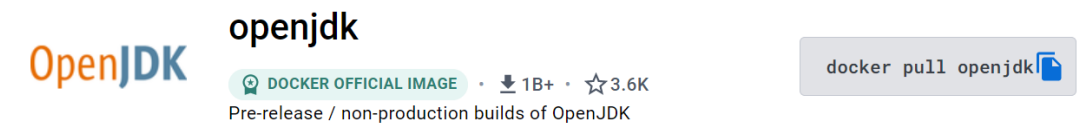
```
[root@ksp ~]# docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
```

З офіційного сайту https://hub.docker.com/ можна завантажити необхідний образ



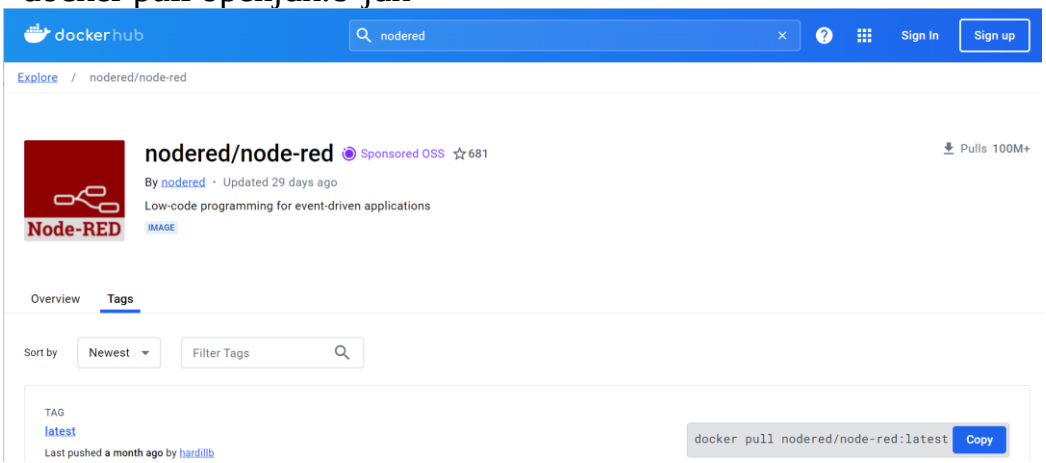






 # встановлюємо образ linux+java, версію openjdk:8-jdk

docker pull openjdk:8-jdk



встановлюємо образ node-red версію latest

docker pull nodered/node-red:latest

 # перевіряємо список образів images

docker images

```
[root@ksp ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
openjdk      8-jdk     b273004037cc   9 months ago    526MB
```

```
[root@ksp ~]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED       SIZE
nodered/node-red    latest    146b0310a475   4 weeks ago   558MB
```

 # перевіряємо список запущених контейнерів

docker ps

 # список усіх контейнерів

docker ps -a

```
[root@ksp ~]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ksp ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

 # створення/запуск нового контейнера **java1** *з існуючого image*

docker run --name java1 -t -i openjdk:8-jdk

```
[root@ksp ~]# docker run --name java1 -t -i openjdk:8-jdk
root@414eb503d276:/#
```

 # створення/запуск нового контейнера **java8** *з існуючого image*

docker run --name java8 -t -i -p 8282:8080 openjdk:8-jdk

-t термінал всередині нового контейнера

-p відповідні порти:

- 8080 внутрішній порт у контейнері
- порт 8282 на машині Linux

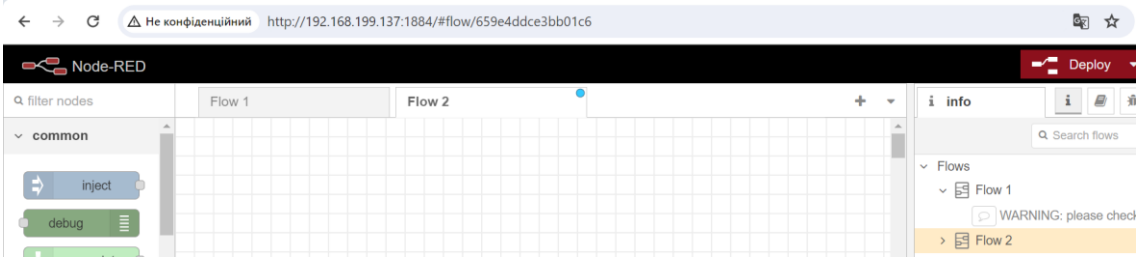
```
[root@ksp ~]# docker run --name java8 -t -i -p 8282:8080 openjdk:8-jdk
root@f008d98d9fff:/#
```

створення/запуск нового контейнера jnode-red1 з існуючого image

docker run --name node-red1 -t -i -p 1884:1880 nodered/node-red

- 1880 внутрішній порт у контейнері
- порт 1884 на машині Linux

запуск через браузер контейнера node-red1 на віртуальній машині



 # перевіряємо встановлену версію java

```
root@414eb503d276:/# java -version
openjdk version "1.8.0_342"
OpenJDK Runtime Environment (build 1.8.0_342-b07)
OpenJDK 64-Bit Server VM (build 25.342-b07, mixed mode)
```

 # виходимо з контейнера 414eb503d276

exit

 # перевіряємо список запущених контейнерів

docker ps

```
[root@ksp ~]# docker ps
CONTAINER ID   IMAGE              COMMAND             CREATED          STATUS
               PORTS                                        NAMES
39968bae6e79   nodered/node-red   "./entrypoint.sh"   41 minutes ago   Up 4 minu
tes (healthy)   0.0.0.0:1884->1880/tcp, :::1884->1880/tcp   node-red1
```

 # список усіх контейнерів

docker ps -a

```
[root@ksp ~]# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[root@ksp ~]# docker ps -a
CONTAINER ID   IMAGE           COMMAND   CREATED          STATUS
           PORTS     NAMES
414eb503d276   openjdk:8-jdk   "bash"    5 minutes ago    Exited (0) 31 seco
nds ago              java1
f008d98d9fff   openjdk:8-jdk   "bash"    32 minutes ago   Exited (137) 11 mi
nutes ago            java8
```

 # запускаємо наявний контейнер командою docker start <id>

docker start f008d98d9fff

```
[root@ksp ~]# docker start f008d98d9fff
f008d98d9fff
[root@ksp ~]# docker ps
CONTAINER ID   IMAGE           COMMAND   CREATED          STATUS          PO
RTS                                      NAMES
f008d98d9fff   openjdk:8-jdk   "bash"    39 minutes ago   Up 15 seconds   0.
0.0.0:8282->8080/tcp, :::8282->8080/tcp   java8
```

 # зупиняємо контейнер командою docker stop <id>

docker stop f008d98d9fff

 # виконання bash на запущеному контейнері

docker exec -it java8 bash

 # виходимо з контейнера

exit

```
[root@ksp ~]# docker exec -it node-red1 bash
39968bae6e79:~$ exit
exit
[root@ksp ~]#
```

 # створити новий образ командою docker commit <id> image-nodered1 з існуючого

docker commit 39968bae6e79 image-nodered1

 # перевіряємо список образів

docker images

```
[root@ksp ~]# docker commit 39968bae6e79 image-nodered1
sha256:3aaa2b45cc429d9e62242fb01888a3c3af6feb3581815e23eb9ed0c207e7327b
[root@ksp ~]# docker images
REPOSITORY         TAG       IMAGE ID       CREATED          SIZE
image-nodered1     latest    3aaa2b45cc42   56 seconds ago   558MB
nodered/node-red   latest    146b0310a475   4 weeks ago      558MB
openjdk            8-jdk     b273004037cc   21 months ago    526MB
```

 # створення/запуск нового контейнера **node-red2** *з існуючого* image-nodered1

docker run --name node-red2 -t -i image-nodered1

```
[root@ksp ~]# docker run --name node-red2 -t -i image-nodered1
10 May 10:16:58 - [info]
```

 # перевіряємо

docker ps –a

```
[root@ksp ~]# docker ps -a
CONTAINER ID   IMAGE                 COMMAND             CREATED             STATUS
                        PORTS     NAMES
4b17910527ef   image-nodered1        "./entrypoint.sh"   4 minutes ago       Exited
 (130) 43 seconds ago             node-red2
39968bae6e79   nodered/node-red      "./entrypoint.sh"   About an hour ago   Exited
 (143) 18 minutes ago             node-red1
```

видалити контейнер командою docker rm <id>

docker rm 4b17910527ef

```
[root@ksp ~]# docker rm 4b17910527ef
4b17910527ef
[root@ksp ~]# docker ps -a
CONTAINER ID   IMAGE                 COMMAND             CREATED             STATUS
                        PORTS     NAMES
39968bae6e79   nodered/node-red      "./entrypoint.sh"   About an hour ago   Exited
 (143) 20 minutes ago             node-red1
```

видалити образ командою docker rmi <id>

docker rmi 3aaa2b45cc42

```
[root@ksp ~]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED             SIZE
image-nodered1      latest    3aaa2b45cc42   56 seconds ago      558MB
nodered/node-red    latest    146b0310a475   4 weeks ago         558MB
openjdk             8-jdk     b273004037cc   21 months ago       526MB
```

Посилання на інформаційні сайти

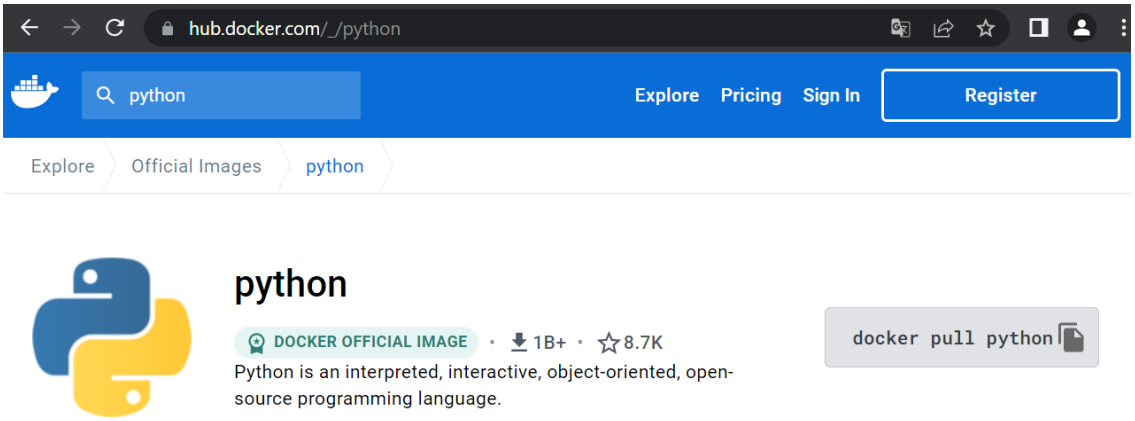
- https://www.baeldung.com/linux/assign-port-docker-container
- https://docs.docker.com/engine/reference/builder/
- https://docs.docker.com/get-started/

3. **Створення нового образу та контейнера за допомогою Dockerfile**

Файл Dockerfile

```
# syntax=docker/dockerfile:1
 FROM python:latest
ADD server.py /server/
ADD index.html /server/
WORKDIR /server/

#RUN python ./server.py
CMD ["python", "./server.py"]
EXPOSE 1234
```



Усі Dockerfile починаються з команди **FROM**. Вона визначає базовий образ для початку процесу побудови контейнера. Це може бути будь-який образ, в тому числі і створені вами до цього. Якщо вказаний вами образ не

знайдений на хості, Докер спробує знайти і завантажити його. Ця команда в Dockerfile завжди повинна бути вказана першою.

Команда **ADD** бере два аргументи, шлях звідки скопіювати файл і шлях куди скопіювати файли у власну файлову систему контейнера. Якщо ж source шляхом є URL (тобто адреса веб-сторінки) - то вся сторінка буде скачана і поміщена в контейнер.

Команда **EXPOSE** повідомляє Docker, що контейнер прослуховує вказані мережеві порти під час виконання.

Команда **RUN** є основною командою для виконання команд під час написання **Dockerfile**. Вона бере команду як аргумент і запускає її з образу. На відміну від CMD, цю команду використовують для побудови образу (можна запустити кілька RUN поспіль, на відміну від CMD). Файл Docker може містити багато кроків RUN, які накладаються один на одного для створення образу.

CMD - це команда, яку контейнер виконує за замовчуванням, коли ви запускаєте зібраний образ. Файл Docker використовуватиме лише остаточний визначений CMD. CMD можна перевизначити під час запуску контейнера за допомогою

```
docker run $image $other_command
```

Команда **WORKDIR** вказує директорію, з якої буде виконуватися команда CMD.

ENTRYPOINT також тісно пов’язана з CMD і може змінювати спосіб запуску контейнера з образу.

Файл index.html

```
<html>
  <head>
    <title>Hello from Ldubgd</title>
  </head>
  <body>
    <h1>This is simple page</h1>
     <br/>
     <h2>Hello world!</h2>
  </body>
</html>
```

Файл server.py

```
#!/usr/bin/env python3
 import http.server
 import socketserver
 handler = http.server.SimpleHTTPRequestHandler
 with socketserver.TCPServer(("", 1234), handler) as httpd:
 httpd.serve_forever()
```

3.1 Створюємо архів **server.zip** з трьох файлів
Dockerfile
index.html
server.py

3.2 Копіюємо файл **server.zip з cmd** в папку root нашої віртуальної машини за допомогою команди

scp server.zip root@192.168.199.137:/root/

```
D:\K_СП\Лабораторні\Taska Docker>scp  server.zip root@192.168.199.137:/root/
root@192.168.199.137's password:
server.zip                                                    100%  718     0.7KB/s   00:00
```

 # перевіряємо наявність файлу server.zip в папці root

ls

```
[root@ksp ~]# ls
2.3.0.2-compiled.zip  index.html    opencart
anaconda-ks.cfg       index.html.1  server.zip
```

 # створюємо папку server

mkdir server

 # переносимо server.zip файл у папку server

mv server.zip server/

 # заходимо в папку server

cd server

```
[root@ksp ~]# mkdir server
[root@ksp ~]# ls
2.3.0.2-compiled.zip  index.html    opencart  server.zip
anaconda-ks.cfg       index.html.1  server
[root@ksp ~]# mv server.zip server/
[root@ksp ~]# cd server
[root@ksp server]# ls
server.zip
```

#розархівовуємо

unzip server.zip

```
[root@ksp server]# unzip server.zip
Archive:  server.zip
  inflating: server/Dockerfile
  inflating: server/index.html
  inflating: server/server.py
```

 # заходимо в папку server

cd server

```
[root@ksp server]# cd server
[root@ksp server]# ls
Dockerfile  index.html  server.py
[root@ksp server]# pwd
/root/server/server
```

 # створити новий образ

docker build -t serverimage .

-t flag tags нашого image

 # перевіряємо список образів images

docker images

```
[root@ksp server]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED              SIZE
serverimage         latest    befce442b333   About a minute ago   1.02GB
nodered/node-red    latest    146b0310a475   4 weeks ago          558MB
openjdk             8-jdk     b273004037cc   21 months ago        526MB
```

 # створення/запуск нового контейнера **myserver** *з існуючого*

docker run --name myserver -t -i -p 1234:1234 serverimage

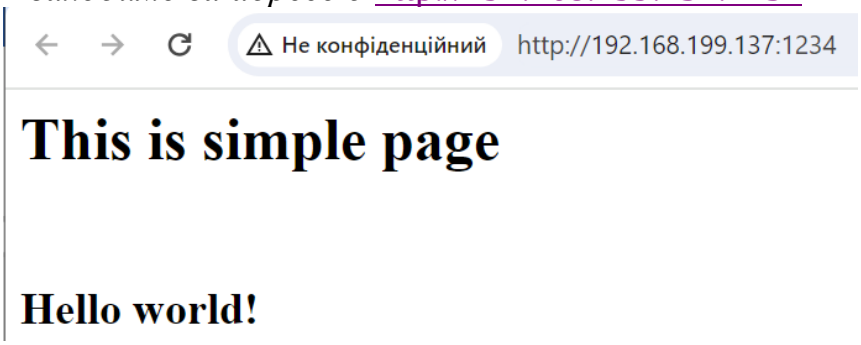
-p 1234:1234 – зіставляє порт 1234 хоста з портом 1234 у контейнері

```
[root@ksp server]# docker ps -a
CONTAINER ID   IMAGE             COMMAND                  CREATED             ST
ATUS                         PORTS     NAMES
ebebfada4c1a   serverimage       "python ./server.py"     About a minute ago   Ex
ited (130) 45 seconds ago              myserver
39968bae6e79   nodered/node-red  "./entrypoint.sh"        3 hours ago          Ex
ited (143) About an hour ago           node-red1
```

 # запускаємо наявний контейнер командою

docker start ebebfada4c1a

 # заходимо за адресою http://192.168.199.137:1234



#Наш контейнер відпрацював

4. Завдання на виконання.

1. Ознайомитися з теоретичними матеріалом по лабораторній роботі.
2. Встановити docker з сайту docker.com https://get.docker.com/
3. Використовуйте базовий образ із центру докерів, наприклад: Alpine Linux, node-red або openjdk;
4. Створіть новий контейнер із завантаженого images;
5. Запустіть створений контейнер;
6. Зупиніть створений контейнер;
7. Створіть новий образ з існуючого;
8. Видаліть контейнер;
9. Розробіть Dockerfile для створення зображення та контейнера на основі Apache HTTP.
10. Додати рівень з Apache HTTP Server (необхідно встановити HTTP Server).
11. Змінити файл index.html за замовчуванням. Використовуйте просту веб-сторінку, наприклад <h1>Hello World!</h1>.
12. Створіть новий образ з використанням Dockerfile.
13. Запуск контейнера за допомогою HTTP-сервера Apache.
14. В процесі виконання робити скріншоти для оформлення звіту.

Підготувати звіт про виконання лабораторної роботи та зробити висновки.

4. Результати виконання роботи та оформлення звіту

Зміст звіту:

1) титульний аркуш;
2) короткі теоретичні відомості;

3) скріншоти з виконаними завданнями та висновки.

Контрольні питання
1. Що таке контейнер, яке його призначення?
2. У чому полягають відмінності між віртуальними машинами та контейнерами?
3. У чому полягають переваги та недоліки контейнерів?
4. Які технології використовують під час створення контейнерів?
5. Що таке образ контейнера та з чого він складається?
6. Що таке репозиторій?
7. Які основні компоненти Docker?
8. Що таке реєстри? Яке їх призначеня?
9. Які вам відомі інструментальні засоби для Docker?
10. Що таке Dockerfile?
11. Які команди використовують для роботи з реєстрами?
12. Що таке рівні образу контейнера?
13. Які вам відомі команди щодо керування контейнерами?
14. Як створити образ контейнера?
15. Як переглянути активні та неактивні контейнери?
16. Які команди використовують під час роботи з Dockerfile?
17. У чому полягає життєвий цикл програмного забезпеченні з використанням контейнера Docker?
18. Як запустити контейнер Docker за допомогою створеного образу?

Методичну розробку склав
доцент кафедри інформаційних технологій
та систем електронних комунікацій Юрій БОРЗОВ