Опис завдання:

1. Створити контейнери з кожним із типів мережі:
   1.1 Bridge
   1.2 Host
   1.3 None
   1.4 Macvlan (зі Staytic IP та DHCP)
2. Створити 2 контейнери налаштовані на використання суміжного Docker Volume

Результат завдання:

1. Завантажити на сервер файл з командами використаними для створення контейнерів з різними типами мережі
2. Завантажити на сервер файл з командами використаними для демонстрації функцій кожного типу мережі
3. Завантажити на сервер файл з командами використаними для створення файлу в одному з контейнерів та демонстрації його вмісту на другому контейнері

План

1. Створюємо контейнер з типом мережі Bridge
2. Створюємо контейнер з типом мережі Host
3. Створюємо контейнер з типом мережі None
4. Створюємо контейнери з типом мережі Macvlan(Staytic IP + DHCP)
5. Створюємо 2 контейнери налаштовані на використання суміжного Docker Volume

Створити контейнери з кожним із типів мережі (Bridge, Host, None, Macvlan (зі Staytic IP та DHCP))

1.Створюємо контейнер з типом мережі Bridge

Команда:

```
sudo docker run -d --name bridge_container busybox sleep 3600
```

Відгук:

```
c3eaf3843071bc71066c2cd8abbfdd4ca69d147512afa62754d278e679e44b3c
```

Контейнер створено. Треба зауважити, шо тип мережі bridge використовується за замовченням і якшо явно не вказати тип мережі, то буде використано саме bridge. Перевіримо чи працює контейнер

Команда:

```
sudo docker ps
```

Відгук:

```
CONTAINER ID   IMAGE     COMMAND        CREATED          STATUS
PORTS     NAMES
```

```
c3eaf3843071   busybox   "sleep 3600"   24 seconds ago   Up 23
seconds             bridge_container
```

Контейнер працює. Тепер створимо bridge мережу для взаємодії двох нових контейнерів якім призначимо цю мережу

Команда:

```
sudo docker network create my_bridge_network
```

Відгук:

```
e692885c55649e966f332b0a291e5e390b08036a4b0c56e884b028124cb10ddf
```

Дивимося чи з’явилася в нас нова мережа

Команда:

```
sudo docker network ls
```

Відгук:

```
NETWORK ID     NAME                DRIVER    SCOPE
ea4d1bd2fe62   bridge              bridge    local
bdafb993de32   host                host      local
e692885c5564   my_bridge_network   bridge    local
2695dbcfa34b   none                null      local
```

Мережа з’явилася. Слід додати, шо для власної мережі bridge ми можемо задавати серед іншого – Subnet, Gateway, мережевий інтерфейс, DNS (вказані ті параметри з якими на зараз мав справу, звісно шо їх більше, подивитися всі можна командою docker network inspect ім’я_мережі)

Створимо два нових контейнера і призначмо їм новостворену мережу

Команда:

```
sudo docker run -d --name container1 --network
my_bridge_network busybox sleep 3600
```

Відгук:

```
e5ffdbfa852706a02db6970d36a295b1541ed21b1e5dfb27622a64e0b05320fe
```

Команда:

```
sudo docker run -d --name container2 --network
my_bridge_network busybox sleep 3600
```

Відгук:

```
808259b9e74be3d17330dac340065173c595b2df16d299494a2c4df23789148a
```

Команда:

```
sudo docker ps
```

Відгук:

```
CONTAINER ID   IMAGE              COMMAND
CREATED             STATUS                    PORTS     NAMES
808259b9e74b   busybox            "sleep 3600"            About
a minute ago   Up About a minute                 container2
e5ffdbfa8527   busybox            "sleep 3600"            About
a minute ago   Up About a minute                 container1
```

Контейнери створено і вони працюють

Перевіряємо зв’язок між контейнерами

Команда:

```
sudo docker exec -it container1 ping container2
```

Відгук:

```
PING container2 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.148 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.149 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.140 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.214 ms
```

Команда:

```
sudo docker exec -it container2 ping container1
```

Відгук:

```
PING container1 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.061 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.181 ms
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.162 ms
64 bytes from 172.18.0.2: seq=3 ttl=64 time=0.178 ms
```

Зв’язок між контейнерами є

2.Створюємо контейнер з типом мережі Host

При використанні цього типу мережі стек контейнера не ізольований від хоста Docker. Якщо контейнер прив'язується до 80 порту (наприклад, nginx), він буде доступний порту 80 IP-адресу хоста

Команда:

```
sudo docker run -d --name host_container --network host nginx
```

Відгук:

```
bd63da094c1df2d82eeb2d0c4ea5c07054744b92008c3ea88b339a5ec525aa8c
```

Контейнер створено. Тут ми явно вказали тип мережі host, контейнер з цім типом мережі не має ip адреси та використовує спільні порти хоста на якому працює. Перевірмо його роботу

Команда:

```
sudo docker ps
```

Відгук:

```
CONTAINER ID   IMAGE     COMMAND                  CREATED
STATUS          PORTS     NAMES
bd63da094c1d   nginx     "/docker-entrypoint.…"   2 minutes ago
Up 2 minutes              host_container
```

Контейнер працює добре. Зробимо ще пару тестів

Перед запуском контейнера ми перевірили чи не зайнятий 80 порт на якому працює nginx у цьому контейнері – порт був не зайнятий. Перевірмо порт хоста тепер

Команда:

```
sudo netstat -tulnp | grep :80
```

Відгук:

```
tcp        0      0 0.0.0.0:80              0.0.0.0:*
LISTEN      10937/nginx: master
tcp6       0      0 :::80                   :::*
LISTEN      10937/nginx: master
```

Тепер перевірмо роботу самого веб сервера

Команда:

```
curl http://10.0.2.15
```

Відгук:

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Бачимо, шо сервер відгукається і віддає помилку 404, бо ми не маємо у цьому контейнері стартової сторінки. Але він працює, тож все ок

3.Створюємо контейнер з типом мережі None

Тип мережі None відключає всі мережі. Часто використовується в додаток при опису нестандартних мереж, щоб вимкнути все зайве

Команда:

```
sudo docker run -d --network none
 --name none_container busybox sleep 3600
```

Відгук:

```
185dae982d0b9be4f3cd37f14c38095fc0848d1dde679fa8d5ae56ff56b8e3bb
```

Контейнер створено, тепер перевірмо його роботу

Команда:

```
sudo docker ps
```

Відгук:

```
CONTAINER ID   IMAGE     COMMAND        CREATED          STATUS
PORTS     NAMES
185dae982d0b   busybox   "sleep 3600"   28 seconds ago   Up 28
seconds             none_container
```

Контейнер працює добре

4.Створюємо контейнер з типом мережі Macvlan(Staytic IP + DHCP)

Створення macvlan-мережі зі статичною адресою та підтримкою DHCP

Щоб контейнер отримував IP-адресу через DHCP, потрібно створити macvlan-мережу в режимі bridge і дозволити контейнеру взаємодіяти з DHCP-сервером у фізичній мережі

Команда:

```
sudo docker network create -d macvlan --subnet=192.168.1.0/24
--gateway=192.168.1.1 -o parent=enp0s3 -o macvlan_mode=bridge
macvlan_network
```

Відгук:

```
0a28bfe3a88ddc54cc6de78e37128b0fe69ebe0524d252f6364d5c26770735ac
```

Команда:

```
sudo docker network ls
```

Відгук:

```
NETWORK ID     NAME              DRIVER    SCOPE
d3e727930ba8   bridge            bridge    local
bdafb993de32   host              host      local
0a28bfe3a88d   macvlan_network   macvlan   local
2695dbcfa34b   none              null      local
```

Мережа macvlan з підтримкою DHCP створена

Перевіримо чи можемо ми створити контейнер з використанням цієї мережі

Команда:

```
sudo docker run -d --name macvlan_dhcp --network
macvlan_network busybox udhcpc -i enp0s3
```

Відгук:

```
23616ab12618443e8b54097c976aef2373fc357e7269fbe7ccb85ed2ae6fd2fe
```

Контейнер створено, але ip-адреси він не має, того шо в нас немає DHCP сервера у фізичній мережі

Перевіряємо його роботу

Команда:

```
sudo docker ps -a
```

Відгук:

```
CONTAINER ID   IMAGE     COMMAND                  CREATED
STATUS                      PORTS     NAMES
23616ab12618   busybox   "udhcpc -i enp0s3"   4 minutes ago
Exited (1) 4 minutes ago             macvlan_dhcp
```

Контейнер створився але певно не працює. Спробуємо перевірити чи отримав він адресу

Команда:

```
sudo docker exec -it macvlan_dhcp ifconfig enp0s3
```

Відгук:

```
Error response from daemon: container
23616ab12618443e8b54097c976aef2373fc357e7269fbe7ccb85ed2ae6fd2fe
is not running
```

Схоже таки те, шо відсутній DHCP сервер повпливав на цю ситуацію. Перезапуск контейнера ситуацію не змінює

Спробуємо створити два нових контейнера використовуючи цю саму мережу (macvlan_network) з якою не запрацював наш dhcp контейнер. Але цього разу призначимо самостійно ip-адреси контейнерам

Команда:

```
sudo docker run -d --name macvlan_container --network
macvlan_network --ip 192.168.1.100 busybox sleep 3600
```

Відгук:

```
6ffb79f127f9b402f00fd037cf4078d17310febad5f25e316d267bcc8193a829
```

Команда:

```
sudo docker run -d --name macvlan_container2 --network
macvlan_network --ip 192.168.1.101 busybox sleep 3600
```

Відгук:

```
ea9811786527003a20928362d5676c1c8b497262bb2c6b51e76885bbe45e81cf
```

Перевірмо чи працюють новостворені контейнери

Команда:

```
sudo docker ps -a
```

Відгук:

```
CONTAINER ID   IMAGE     COMMAND              CREATED
STATUS                     PORTS     NAMES
ea9811786527   busybox   "sleep 3600"         5 seconds ago    Up
4 seconds                          macvlan_container2
6ffb79f127f9   busybox   "sleep 3600"         26 seconds ago   Up
25 seconds                         macvlan_container
```

Контейнери працюють добре. Тепер спробуємо про пінгувати їх

Команда:

```
sudo docker exec -it macvlan_container ping 192.168.1.101
```

Відгук:

```
PING 192.168.1.101 (192.168.1.101): 56 data bytes
64 bytes from 192.168.1.101: seq=0 ttl=64 time=0.101 ms
64 bytes from 192.168.1.101: seq=1 ttl=64 time=0.069 ms
64 bytes from 192.168.1.101: seq=2 ttl=64 time=0.075 ms
64 bytes from 192.168.1.101: seq=3 ttl=64 time=0.077 ms
```

Другий контейнер пінгується чудово. Тепер перевірмо перший

Команда:

```
sudo docker exec -it macvlan_container2 ping 192.168.1.100
```

Відгук:

```
PING 192.168.1.100 (192.168.1.100): 56 data bytes
```

```
64 bytes from 192.168.1.100: seq=0 ttl=64 time=0.216 ms
64 bytes from 192.168.1.100: seq=1 ttl=64 time=0.086 ms
64 bytes from 192.168.1.100: seq=2 ttl=64 time=0.072 ms
64 bytes from 192.168.1.100: seq=3 ttl=64 time=0.075 ms
```

І перший контейнер добре пінгується. Зв’язок між контейнерами працює добре

5.Створимо 2 контейнери налаштовані на використання суміжного Docker Volume

Спочатку перевірмо чи є в нас якісь volume

Команда:

```
sudo docker volume ls
```

Відгук:

```
DRIVER    VOLUME NAME
```

Ніяких volume зараз у нас немає

Створюємо два контейнери для вправи

Команда:

```
sudo docker run -d --name volume_container1 -v
my_volume:/data busybox sleep 3600
```

Відгук:

```
97ee0b5ab839a36c5a6614994d70496f432f8f067d2034f61a7cab1e288c98bf
```

Команда:

```
sudo docker run -d --name volume_container2 -v
my_volume:/data busybox sleep 3600
```

Відгук:

```
f6f7a6905f56eb75d2c9522445f0779e80d6b2d218fccddc6eb4c0735b30acf3
```

Контейнери створені, перевірмо тепер наявність volume

Команда:

```
sudo docker volume ls
```

Відгук:

```
DRIVER    VOLUME NAME
local     my_volume
```

З’явився новий volume my_volume. Відмітимо, шо контейнера було створено два, але їм ми вказали використовувати один і той самий volume - my_volume

Тепер створимо файл на my_volume з контейнера volume_container1

Команда:

```
sudo docker exec -it volume_container1 sh
```

Відгук:

```
/ #
*** далі суто команди ***
/ # cd /data
/data # touch testfile
/data # ls
testfile
/data # exit
```

Ми створили файл у теці data і перевірили шо він там є
Тепер повторимо все те саме для контейнера volume_container2
Додатково перевіримо, чи бачимо ми файл створений у контейнері volume_container1

Команда:

```
sudo docker exec -it volume_container2 sh
```

Відгук:

```
/ #
*** далі суто команди ***
/ # cd /data
/data # ls
testfile
/data # touch testfile2
/data # ls
testfile   testfile2
/data # exit
```

Побачили файл зроблений на контейнері volume_container1, додали свій, впевнилися шо бачимо обидва файли і вийшли

Тепер перевірмо, чи бачимо ми створений на volume_container2 файл

Команда:

```
sudo docker exec -it volume_container1 sh
```

Відгук:

```
/ #
*** далі суто команди ***
/ # cd /data
/data # ls
testfile   testfile2
/data # exit
```

Бачимо обидва файли, вправу закінчено

Домашнє завдання до уроку 5:

1.Створити контейнери з кожним із типів мережі:
1.1 Bridge

1.2 Host
1.3 None
1.4 Macvlan (зі Staytic IP та DHCP)
2.Створити 2 контейнери налаштовані на використання суміжного Docker Volume

Студент: Олександр Болотов

Дата виконання завдання: 17.02.2025