Laboratorium 7

Wprowadzenie do Dockera

W sprawozdaniu zamieść zrzuty ekranu z istotnych etapów oraz odpowiedzi na pytania.

Taki symbol oznacza, że trzeba w sprawozdaniu dodać zrzut ekranu (najczęściej 1) z wyniku działania polecenie.

Taki symbol oznacza, że należy dodać opis (najczęściej 1 zdanie) z wyniku działania polecenia.

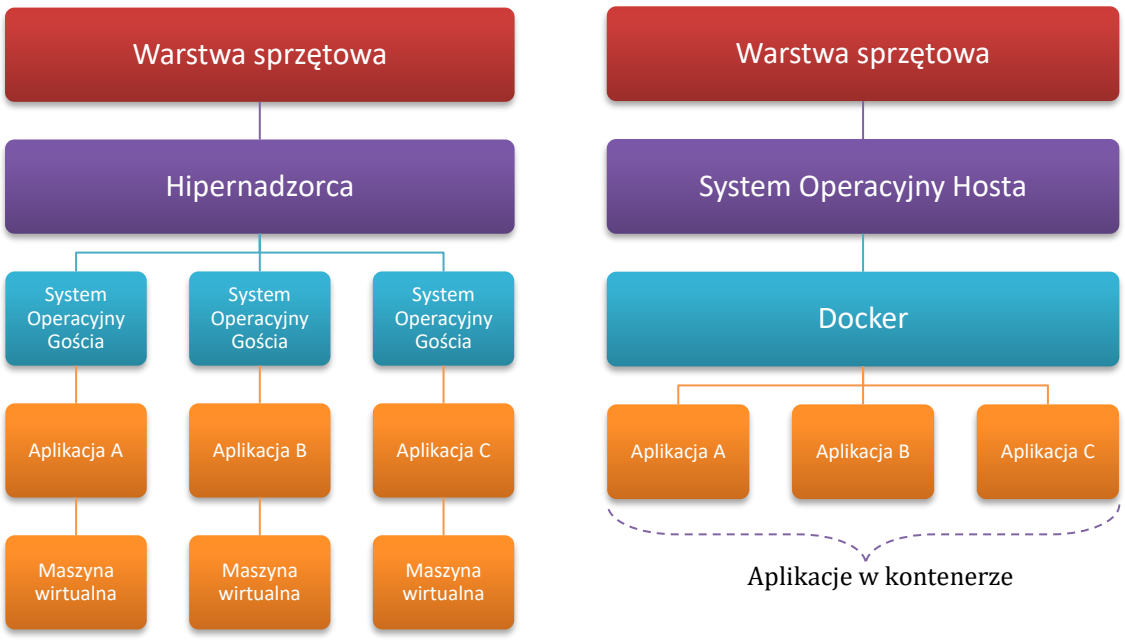
Wstęp

Początki wirtualizacji sięgają lat 60. XX wieku, kiedy IBM opracował system logicznego podziału zasobów komputerów mainframe o nazwie CP/CMS. Mechanizm ten umożliwiał efektywniejsze wykorzystanie zasobów sprzętowych, minimalizując konieczność uruchamiania kolejnych urządzeń. W efekcie ograniczono zużycie energii oraz wprowadzono separację środowisk, dzięki czemu błędy w jednej wirtualnej maszynie nie wpływały na działanie innych instancji.

W 2013 roku, dzięki inicjatywie Solomona Hykesa, powstało nowe narzędzie – Docker. W odróżnieniu od tradycyjnych mechanizmów wirtualizacji, które opierają się na emulacji całego sprzętu komputerowego, Docker wykorzystuje tzw. kontenery. Ta lekka architektura pozwala na efektywniejsze wykorzystanie zasobów systemowych, umożliwiając uruchomienie wielu aplikacji na tej samej infrastrukturze bez symulowania sprzętu. W rezultacie zmniejsza się obciążenie systemu.

W modelu wirtualizacji (po lewej) hipernadzorca, zarządzający procesem, działa bezpośrednio na warstwie sprzętowej. Na nim tworzone są odseparowane maszyny wirtualne. Każda maszyna wirtualna zawiera pełną kopię systemu operacyjnego, aplikacji, niezbędnych plików binarnych i bibliotek – zajmując dziesiątki GB. Maszyny wirtualne mogą się również wolno uruchamiać..

Z kolei Docker działa w ramach infrastruktury, na której zainstalowany jest system operacyjny hosta (np. Linux). Wykorzystując jądro hosta, Docker tworzy odseparowane środowiska dla uruchamianych aplikacji. Dzięki temu, że nie ma potrzeby wirtualizacji komponentów systemu gościa, możliwe jest znaczne ograniczenie zużycia zasobów przy uruchamianiu tych samych aplikacji.



Rysunek 1. Porównanie wirtualizacji i konteneryzacji.*

* https://www.docker.com/resources/what-container/

I. Dockerfile

Jednym z podstawowych elementów Dockera są pliki Dockerfile, które zawierają polecenia używane do budowy obrazów. Zanim przejdziemy do tworzenia obrazu, zainstalujmy Dockera.

1. Uruchom maszynę wirtualną opartą na Debianie.

> Tym razem zainstaluj nakładadkę graficzną!

2. Otwórz terminal.

3. Zaktualizuj listę pakietów:

```
sudo apt update
```

4. Zainstaluj Dockera:

```
sudo apt install -y docker.io
```

5. Sprawdź wersję Dockera: **(zadanie na 3.0)**

```
sudo docker --version
```

6. Aby używać Dockera bez konieczności wpisywania `sudo`, dodaj użytkownika do grupy `docker`:

```
sudo usermod -aG docker $USER
```

7. Uruchom ponownie maszynę wirtualną, aby zastosować zmiany.

8. Otwórz edytor `nano` i utwórz plik `Dockerfile`:

```
nano Dockerfile
```

9. Wypełnij plik następującym kodem:

```
FROM alpine:3.23.2
RUN apk update && apk add --no-cache \
    bash \
    curl
RUN adduser -D limited -s /bin/bash
USER limited:limited
WORKDIR /app
COPY . /app
```

Pierwsze polecenie `FROM alpine:3.23.2` wskazuje obraz bazowy, który będzie podstawą do budowy kolejnych warstw. `Alpine Linux` to minimalistyczna dystrybucja Linuksa zaprojektowana z myślą o bezpieczeństwie, prostocie i wydajności zasobów – kontenery oparte na Alpine uruchamiają się błyskawicznie i zajmują bardzo mało miejsca na dysku oraz w pamięci RAM.

Polecenie `RUN apk update && apk add --no-cache bash curl` odpowiada za:

- aktualizację repozytoriów pakietów,
- instalację `bash` i `curl`.

Te komendy zostaną wykonane podczas budowy obrazu.Kolejne polecenia:

- `RUN adduser -D limited -s /bin/bash` – tworzy nowego użytkownika o nazwie `limited`,
- `USER limited:limited` – przełącza kontekst na konto tego użytkownika.

Na końcu:

- `WORKDIR /app` ustawia katalog roboczy na `/app`,
- `COPY . /app` kopiuje całą zawartość bieżącego katalogu (na maszynie hosta) do katalogu `/app` w kontenerze.

Każdy obraz Dockera może mieć przypisany dowolny tag – w tym przypadku jest to `3.23.2`.

Warto zwrócić uwagę na tag `latest`, który może być mylący, ponieważ nie zawsze odnosi się do najnowszej wersji oprogramowania. Dlatego zaleca się ręczne określanie wersji, jak w tym przykładzie.

10. Zbuduj obraz Dockera o nazwie `psisk`, nadając mu tag odpowiadający Twojemu numerowi indeksu:

```
docker build . -t psisk:twoj_numer_indeksu
```

```
=> [internal] load build definition from Dockerfile                                                                0.0s
=> => transferring dockerfile: 199B                                                                                0.0s
=> [internal] load metadata for docker.io/library/alpine:3.23.2                                                    2.5s
=> [internal] load .dockerignore                                                                                   0.0s
=> => transferring context: 2B                                                                                     0.0s
=> [1/5] FROM docker.io/library/alpine:3.23.2@sha256:865b95f46d98cf867a156fe4a135ad3fe50d2056aa3f25ed31662dff6da4eb62  1.1s
=> => resolve docker.io/library/alpine:3.23.2@sha256:865b95f46d98cf867a156fe4a135ad3fe50d2056aa3f25ed31662dff6da4eb62  0.0s
=> => sha256:865b95f46d98cf867a156fe4a135ad3fe50d2056aa3f25ed31662dff6da4eb62 9.22kB / 9.22kB                       0.0s
=> => sha256:1882fa4569e0c591ea092d3766c4893e19b8901a8e649de7067188aba3cc0679 1.02kB / 1.02kB                      0.0s
=> => sha256:e7b39c54cdeca0d2aae83114bb605753a5f5bc511fe8be7590e38f6d9f915dad 611B / 611B                           0.0s
=> => sha256:1074353eec0db2c1d81d5af2671e56e00cf5738486f5762609ea33d606f88612 3.86MB / 3.86MB                      0.6s
=> => extracting sha256:1074353eec0db2c1d81d5af2671e56e00cf5738486f5762609ea33d606f88612                           0.5s
=> [internal] load build context                                                                                   2.6s
=> => transferring context: 128.27MB                                                                               2.6s
=> [2/5] RUN apk update && apk add --no-cache     bash     curl                                                    2.8s
=> [3/5] RUN adduser -D limited -s /bin/bash                                                                       0.2s
=> [4/5] WORKDIR /app                                                                                              0.0s
=> [5/5] COPY . /app                                                                                               0.6s
=> exporting to image                                                                                              0.3s
=> => exporting layers                                                                                             0.3s
=> => writing image sha256:a73f01b2f3c32d8f7cb5e294300b1e06619d7c5e311f371b970b584f024c3102                        0.0s
=> => naming to docker.io/library/psisk:112233                                                                     0.0s
```


11. Sprawdź utworzony obraz: **(zadanie na 3.0)**

```
docker images
```

```
u112233@112233:~$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
psisk        112233    a73f01b2f3c3   2 minutes ago    147MB
```

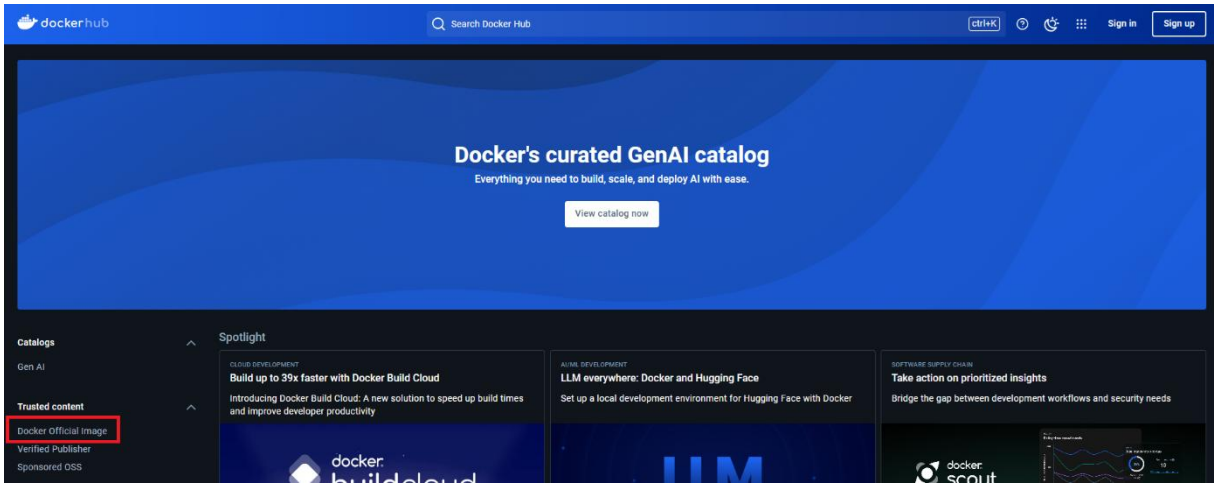
Przed uruchomieniem pierwszego kontenera warto zwrócić uwagę na potencjalne zagrożenia, które mogą się pojawić:

- **Nieaktualne oprogramowanie** – może zawierać znane podatności lub błędy w konfiguracji.
- **Złośliwe oprogramowanie (malware)** – może być umieszczone w obrazie lub pobierane podczas jego uruchamiania.

Aby zwiększyć pewność co do bezpieczeństwa wybranego obrazu, należy go zweryfikować jeszcze przed pobraniem. Najpopularniejszym repozytorium obrazów jest Docker Hub, które oferuje szeroką gamę obrazów. Niestety, nawet tam mogą znajdować się obrazy zawierające błędy bezpieczeństwa. Dlatego zaleca się korzystanie wyłącznie z oficjalnych i zweryfikowanych obrazów.

12. Otwórz przeglądarkę i przejdź do Docker Hub.

13. Filtruj tylko oficjalne obrazy, zaznaczając opcję `DOCKER OFFICIAL IMAGE`.



14. Przejrzyj dostępne obrazy.

Mimo to, jak omawialiśmy na pierwszych zajęciach z utwardzania Linuksa, nawet oficjalne obrazy mogą być zainfekowane. Jednym z rozwiązań jest weryfikacja warstw, z których składa się obraz, oraz przeskanowanie go pod kątem obecności złośliwego kodu za pomocą narzędzi takich jak `Snyk` czy `Trivy`. W tym przypadku skorzystamy z narzędzia `Trivy`.

15. Pobierz skaner `Trivy`:

```
wget https://github.com/aquasecurity/trivy/releases/download/v0.68.2/trivy_0.68.2_Linux-64bit.deb
```

16. Zainstaluj skaner `Trivy`:

```
sudo dpkg -i trivy_0.68.2_Linux-64bit.deb
```

17. Przeskanuj obraz `alpine:3.23.2`: **(zadanie na 3.0)**

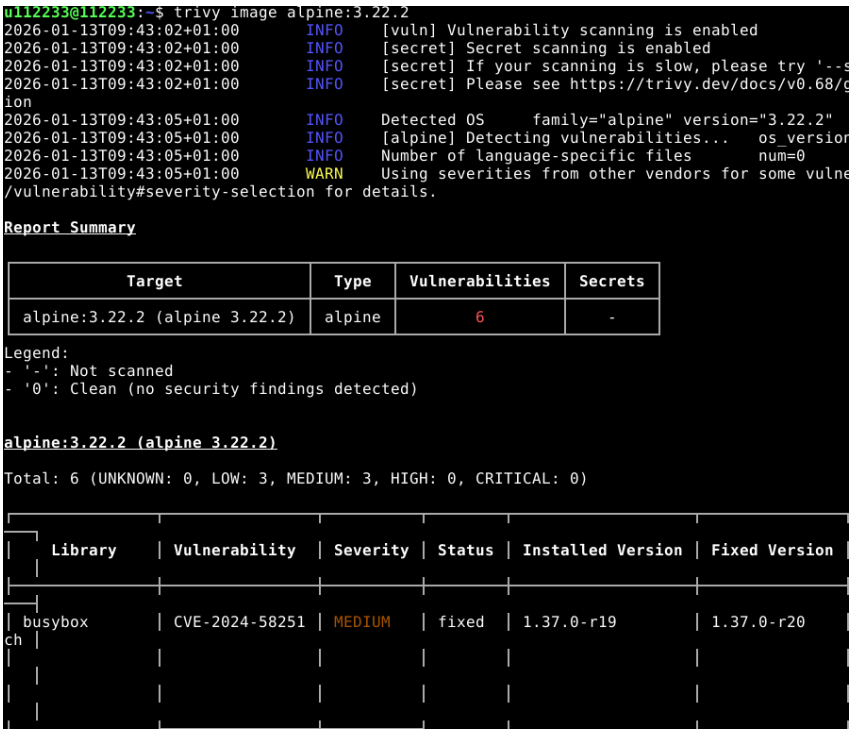
```
trivy image alpine:3.23.2
```

```
u112233@l12233:~$ trivy image alpine:3.23.2
2026-01-13T09:41:39+01:00       INFO    [vulndb] Need to update DB
2026-01-13T09:41:39+01:00       INFO    [vulndb] Downloading vulnerability DB...
2026-01-13T09:41:39+01:00       INFO    [vulndb] Downloading artifact...        repo="mirror.gcr.io/aquasec/trivy-db:2"
80.10 MiB / 80.10 MiB [-------------------------------------------------------------------------------------------] 100.00% 23.17 MiB p/s 3.7s
2026-01-13T09:41:43+01:00       INFO    [vulndb] Artifact successfully downloaded       repo="mirror.gcr.io/aquasec/trivy-db:2"
2026-01-13T09:41:43+01:00       INFO    [vuln] Vulnerability scanning is enabled
2026-01-13T09:41:43+01:00       INFO    [secret] Secret scanning is enabled
2026-01-13T09:41:43+01:00       INFO    [secret] If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2026-01-13T09:41:43+01:00       INFO    [secret] Please see https://trivy.dev/docs/v0.68/guide/scanner/secret#recommendation for faster secret detect
ion
2026-01-13T09:41:46+01:00       INFO    Detected OS     family="alpine" version="3.23.2"
2026-01-13T09:41:46+01:00       WARN    This OS version is not on the EOL list  family="alpine" version="3.23"
2026-01-13T09:41:46+01:00       INFO    [alpine] Detecting vulnerabilities...   os_version="3.23" repository="3.23" pkg_num=16
2026-01-13T09:41:46+01:00       INFO    Number of language-specific files       num=0

Report Summary

┌──────────────────────────┬────────┬─────────────────┬─────────┐
│          Target          │  Type  │ Vulnerabilities │ Secrets │
├──────────────────────────┼────────┼─────────────────┼─────────┤
│ alpine:3.23.2 (alpine 3.23.2) │ alpine │        0        │    -    │
└──────────────────────────┴────────┴─────────────────┴─────────┘

Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)
```

Jak można zauważyć, obecna wersja obrazu wydaje się bezpieczna. Sprawdźmy jednak, jak wyglądała sytuacja w przypadku wcześniejszej wersji, oznaczonej jako `3.22.2`.

18. Porównaj z poprzednią wersją obrazu: **(zadanie na 3.0)**

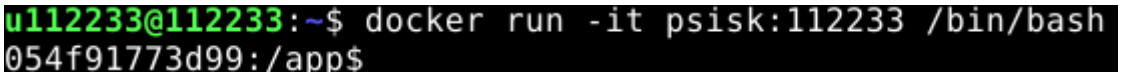
```
trivy image alpine:3.22.2
```



Jak widać, poprzednia wersja obrazu okazała się podatna na zagrożenia. Po zakończeniu weryfikacji możemy teraz przystąpić do uruchomienia kontenera.

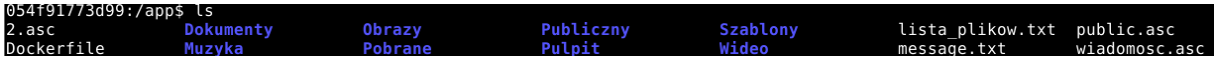
19. Uruchom kontener:

```
docker run -it psisk:twoj_numer_indeksu /bin/bash
```



20. Sprawdź, czy pliki zostały prawidłowo skopiowane: **(zadanie na 3.0)**

```
ls
```



To właśnie największa zaleta Dockera – na Twoim Debianie możesz uruchomić kontener z Alpine, Ubuntu, CentOS czy Fedorą, a one będą działać w izolacji. Twój Debian jest tzw. Hostem. Posiada on jądro Linux (Kernel). Kontener Alpine jest tylko lekką "nakładką" (izolowanym procesem – zupełnie inaczej jak VM'ka), która korzysta z jądra Twojego Debiana, ale posiada własny system plików i menedżer pakietów

Jakie głowne zalety?

- **Izolacja:** Nawet jeśli się coś zepsuje wewnątrz kontenera Alpine (np. uszkodzą się pliki systemowe), Twój Debian pozostanie nienaruszony.
- **Czystość:** Po usunięciu kontenera w systemie Debian nie zostaną żadne ślady po Apache czy plikach Alpine.

21. Wyjdź z kontenera:

```
exit
```

22. Sprawdź, czy kontener został zamknięty (nie działa w tle):

```
docker ps -a
```

```
u112233@112233:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED         STATUS                     PORTS     NAMES
054f91773d99   psisk:112233   "/bin/bash"   8 minutes ago   Exited (0) 9 seconds ago             nostalgic_cori
```

Sprawdźmy jeszcze jeden przykład – własny serwer WWW.

23. Otwórz edytor `nano` i utwórz plik `Dockerfile`:

```
nano Dockerfile
```

24. Wypełnij plik następującym kodem:

```
FROM alpine:3.23.2
RUN apk add --no-cache apache2
CMD ["httpd", "-D", "FOREGROUND"]
```

- `apk add --no-cache apache2`: Instaluje serwer, który automatycznie tworzy domyślny plik `index.html` w folderze `/var/www/localhost/htdocs`.
- `CMD`: Uruchamia serwer tak, by kontener pozostał aktywny.

25. Zbuduj obraz:

```
docker build . -t apache:twoj_numer_indeksu
```

26. Sprawdź utworzony obraz:

```
docker images
```

```
u112233@112233:~$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
apache       112233    b6da66889a5c   8 minutes ago    12.4MB
psisk        112233    a73f01b2f3c3   50 minutes ago   147MB
```

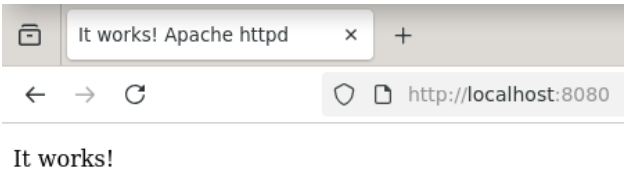
27. Uruchom kontener: **(zadanie na 3.0)**

```
docker run -p 8080:80 apache:twoj_numer_indeksu
```

```
u112233@112233:~$ docker run -p 8080:80 apache:112233
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to s
uppress this message
```

28. Zminimalizuj terminal i uruchom przeglądarkę. Wpisz w pasku adresowym: **(zadanie na 3.0)**

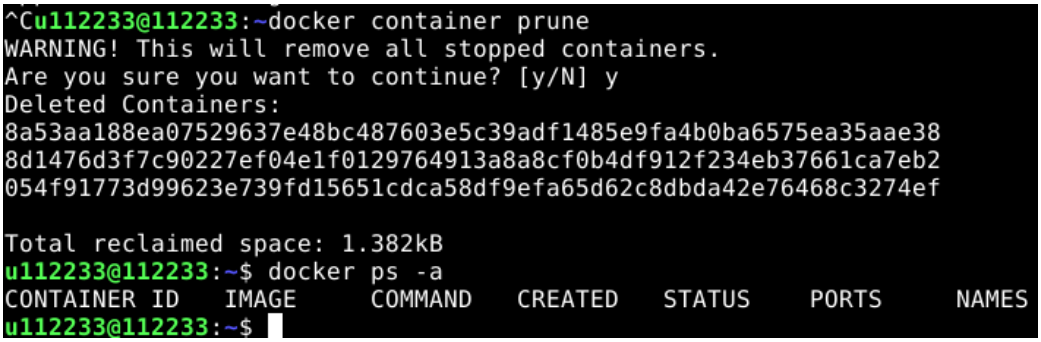
```
http://localhost:8080/
```



29. Wróć do terminala i przerwij działanie kontenera ctrl +c.

30. Usuń wszystkie utworzone kontenery:

```
docker container prune
```



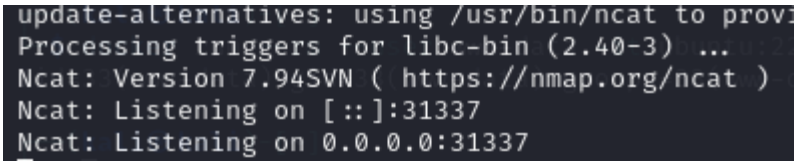
II. Płaska sieć

Administratorzy często nie zmieniają domyślnych ustawień sieciowych (problem ten dotyczy nawet dużych systemów), w wyniku czego kontenery są zwykle konfigurowane w taki sposób, że znajdują się w tym samym segmencie sieci, umożliwiając im wzajemną komunikację. Taki sposób konfiguracji jest standardowy w przypadku domyślnego sterownika sieciowego Dockera. Czy jest to jednak podejście właściwe? To kwestia dyskusyjna. Z pewnością jednak nie jest to optymalne rozwiązanie z perspektywy bezpieczeństwa. Sprawdźmy, czy kontenery rzeczywiście domyślnie mogą się wzajemnie "widzieć".

1. Uruchom trzy instancje terminala.

2. W pierwszym terminalu wpisz następującą komendę, aby utworzyć kontener:

```
docker run --rm -it --name "kali1" kalilinux/kali-rolling bash -c "apt-get update
&& apt-get install -y ncat && ncat -lvp 31337"
```



Powyższa komenda uruchamia nowy kontener na podstawie obrazu Kali Linux w wersji `rolling`, instaluje narzędzie `ncat`, a następnie ustawia je w trybie nasłuchiwania na porcie `31337`.

3. Przejdź do drugiego terminala. Zainstaluj narzędzie jq (do przetwarzania danych w formacie JSON):

```
sudo apt install jq -y
```

4. Wykonaj weryfikację adresu IP kontenera uruchomionego w domyślnej sieci bridge:

```
docker inspect bridge | jq '.[0].Containers'
```

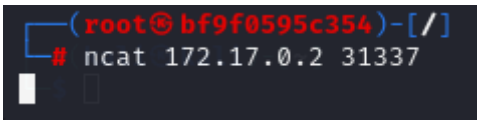
```
u112233@112233:~$ docker inspect bridge | jq '.[0].Containers'
{
  "64094b1c4587783bf2d2db7647bc0f87cdb912eecb1467353fa7a07731282176": {
    "Name": "kali1",
    "EndpointID": "afc24f28e153ec2732b029d4c1752a038d5fcb2743cd8c415429e2ac7678afd4",
    "MacAddress": "02:42:ac:11:00:02",
    "IPv4Address": "172.17.0.2/16",
    "IPv6Address": ""
  }
}
```

5. Przejdź do trzeciego terminala i uruchom kolejny kontener:

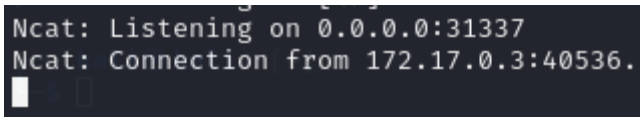
```
docker run --rm -it --name "kali2" kalilinux/kali-rolling bash -c "apt-get update && apt-get install -y ncat && bash"
```

6. Nawiąż połączenie z drugim kontenerem przy użyciu narzędzia `ncat`:

```
ncat adres_ip_pierwszego_kontenera port
```



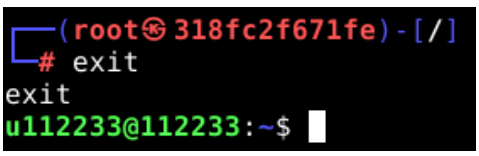
7. Upewnij się, że komunikacja faktycznie nastąpiła. Przejdź do pierwszego terminala. **(zadanie na 3.5)**



Udowodniliśmy, że domyślnie Docker konfiguruje sieć w taki sposób, aby kontenery były w tym samym segmencie. Ograniczmy takie zachowanie poprzez umieszczenie kontenerów w osobnych podsieciach.

8. Przejdź do trzeciego terminala i przerwij komunikację kombinacją klawiszy `Ctrl + C`.

9. W tym samym terminalu wyjdź z kontenera komendą `exit`.



10. Następnie utwórz dwie nowe sieci typu most:

```
docker network create net-kali1
docker network create net-kali2
```

Teraz należy uruchomić kontenery, wskazując, w jakiej sieci mają się znaleźć.

11. Przejdź do pierwszego terminala i uruchom następującą komendę:

```
docker run --rm -it --name "kali1" --network "net-kali1" kalilinux/kali-rolling bash
-c "apt-get update && apt-get install -y ncat && ncat -lvp 31337"
```

Jak widzisz, kontener ponownie przeszedł w tryb nasłuchiwania.

12. Przejdź do drugiego terminala i uruchom drugi kontener:

```
docker run --rm -it --name "kali2" --network "net-kali2" kalilinux/kali-rolling bash
-c "apt-get update && apt-get install -y ncat && bash"
```

13. Przejdź do terminala numer 3 i wykonaj poniższe komendy, aby sprawdzić, w jakich podsieciach znajdują się kontenery: **(zadanie na 4.0)**

```
docker inspect net-kali1 | jq '.[0].Containers'
docker inspect net-kali2 | jq '.[0].Containers'
```

```
u112233@112233:~$ docker inspect net-kali1 | jq '.[0].Containers'
docker inspect net-kali2 | jq '.[0].Containers'
{
  "95211c11dbcfe1a649da918cfd7d69056641614f9820f3ce01759f7d0d4c8ad7": {
    "Name": "kali1",
    "EndpointID": "28ef835c0820b15712b4840659e61bf0a99f261ff9d6b8d86c29d79bce44996e",
    "MacAddress": "02:42:ac:12:00:02",
    "IPv4Address": "172.18.0.2/16",
    "IPv6Address": ""
  }
}
{
  "48479b4cf5e68ad0a516747d9a654e95938a0935d9014b41c1cb64c1250be376": {
    "Name": "kali2",
    "EndpointID": "4b84820ed1abad5d37e2ae93d54aeec4a72df4de3283ee51b2b2981021916d7f",
    "MacAddress": "02:42:ac:13:00:02",
    "IPv4Address": "172.19.0.2/16",
    "IPv6Address": ""
  }
}
```

Znający podstawy sieci komputerowych już teraz widzą, że komunikacja między kontenerami jest niemożliwa. Mimo to, upewnijmy się, że tak jest...

14. Przejdź do drugiego terminala i spróbuj nawiązać komunikację z kontenerem `kali1`, podobnie jak w punkcie 2.8. Komunikacja nie powinna być możliwa.

```
┌──(root㉿3456e5f71d61)-[/]
└─# ncat 172.18.0.2 31337
Ncat: TIMEOUT.
```

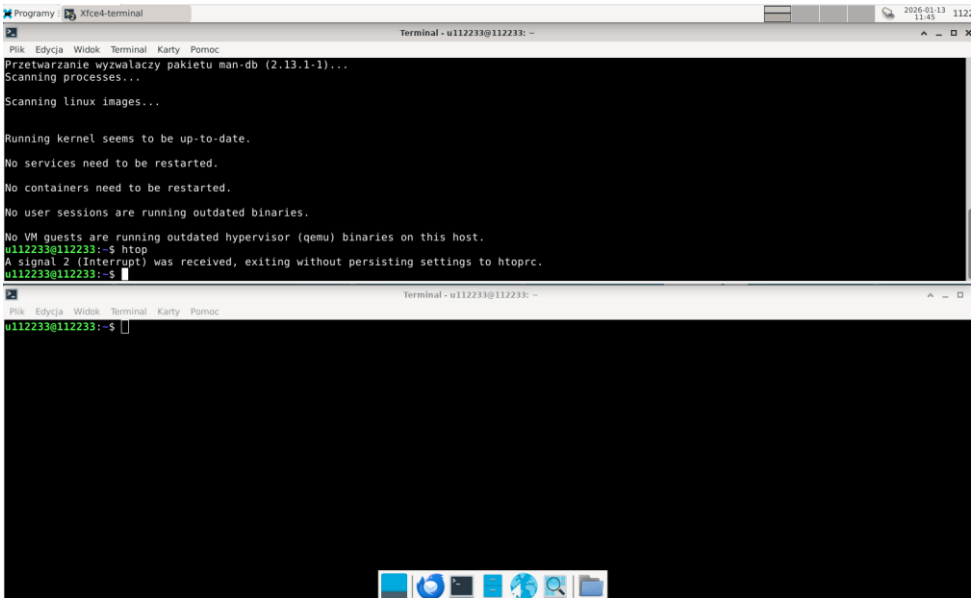
III. Brak ograniczeń zasobów

Wyobraźmy sobie sytuację, w której kontener z aplikacją zostaje zaatakowany, a napastnik uzyskuje dostęp do niego. Jeśli dostęp do zasobów nie został określony, kontener będzie mógł wykorzystać wszystkie zasoby pamięciowe/CPU hosta. Zasymulujmy taką sytuację, używając tzw. zip bomby. Atak polega na rozpakowaniu odpowiednio przygotowanego pliku `.zip`, co powoduje zajęcie całej dostępnej pamięci operacyjnej, procesora lub przestrzeni dyskowej.

1. Wyjdź z kontenerów i zamknij wszystkie terminale.

2. Sprawdź, czy masz uruchomione jakieś kontenery (ppkt 1.22 oraz 1.30). Jeśli tak, zatrzymaj je i usuń.

3. Uruchom 2 instancje terminala i ustaw je jeden nad drugim.

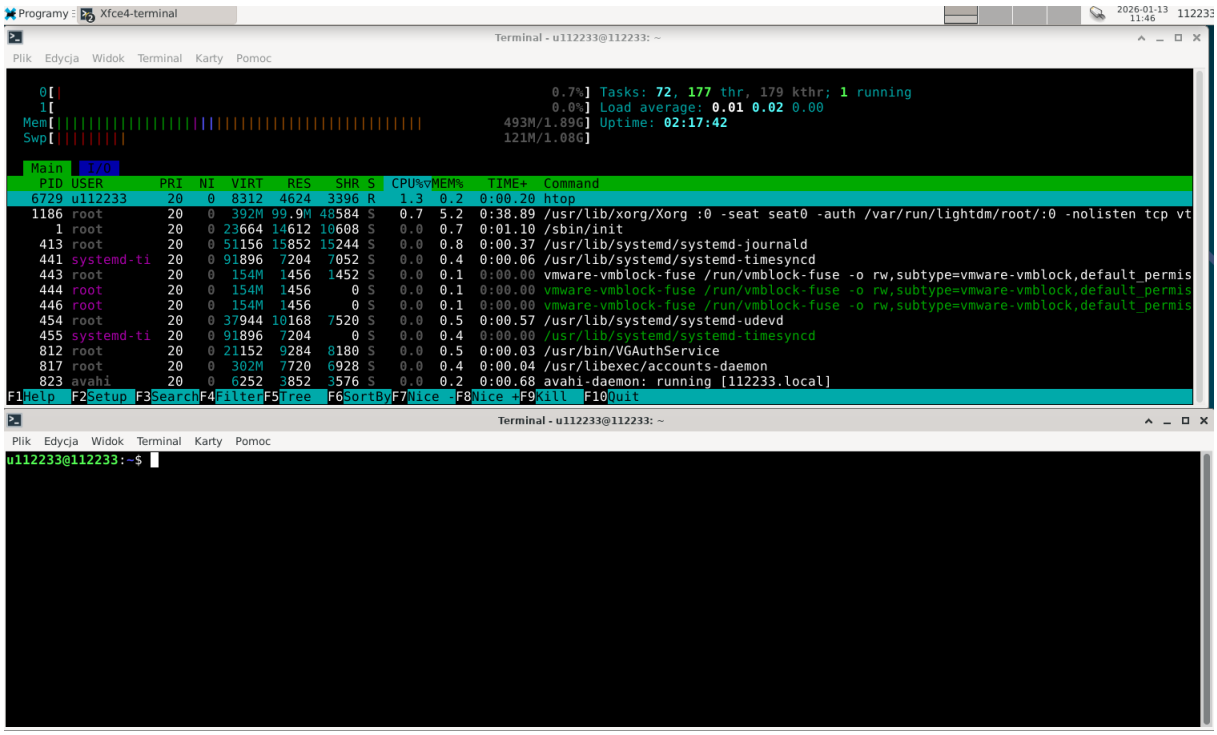


4. Zainstaluj monitor zasobów `htop`:

```
sudo apt install htop
```

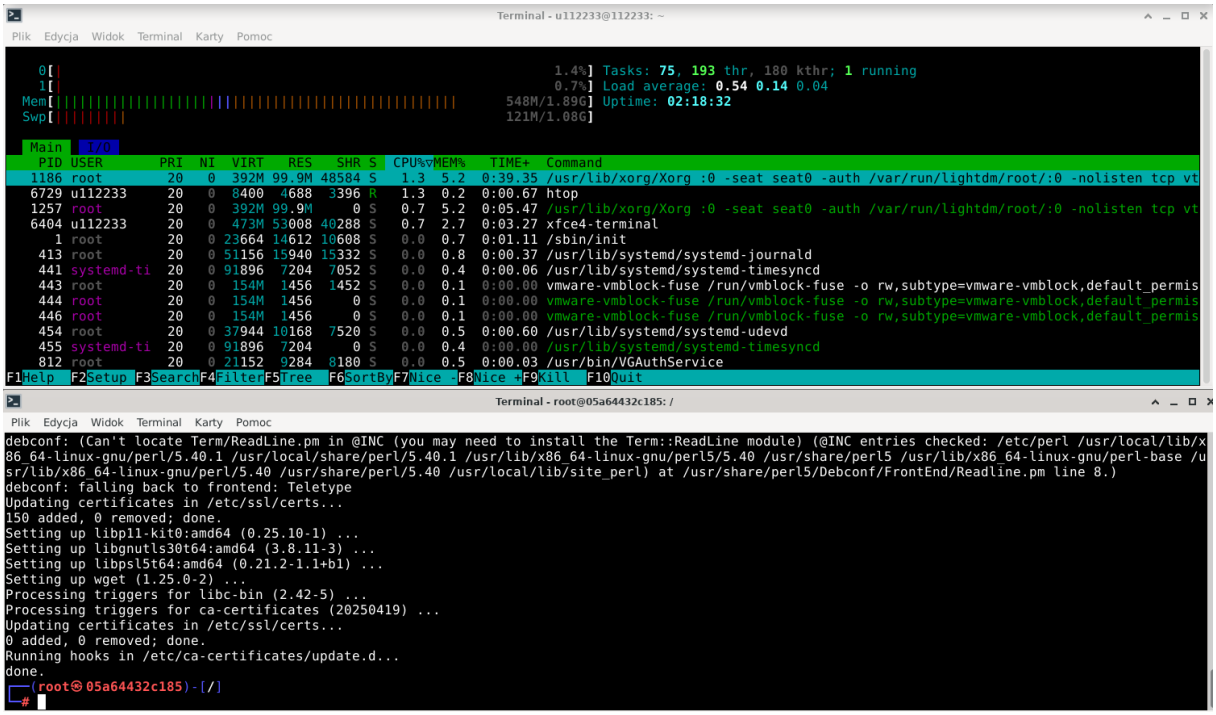
5. W górnej części terminala uruchom `htop`:

```
htop
```



6. W dolnej części uruchom kontener z Kali Linux:

```
docker run --rm -it kalilinux/kali-rolling bash -c "apt-get update && apt-get install -y p7zip-full wget && bash"
```



7. Pobierz zip bombę w kontenerze:

```
wget https://www.bamsoftware.com/hacks/zipbomb/zbsm.zip
```

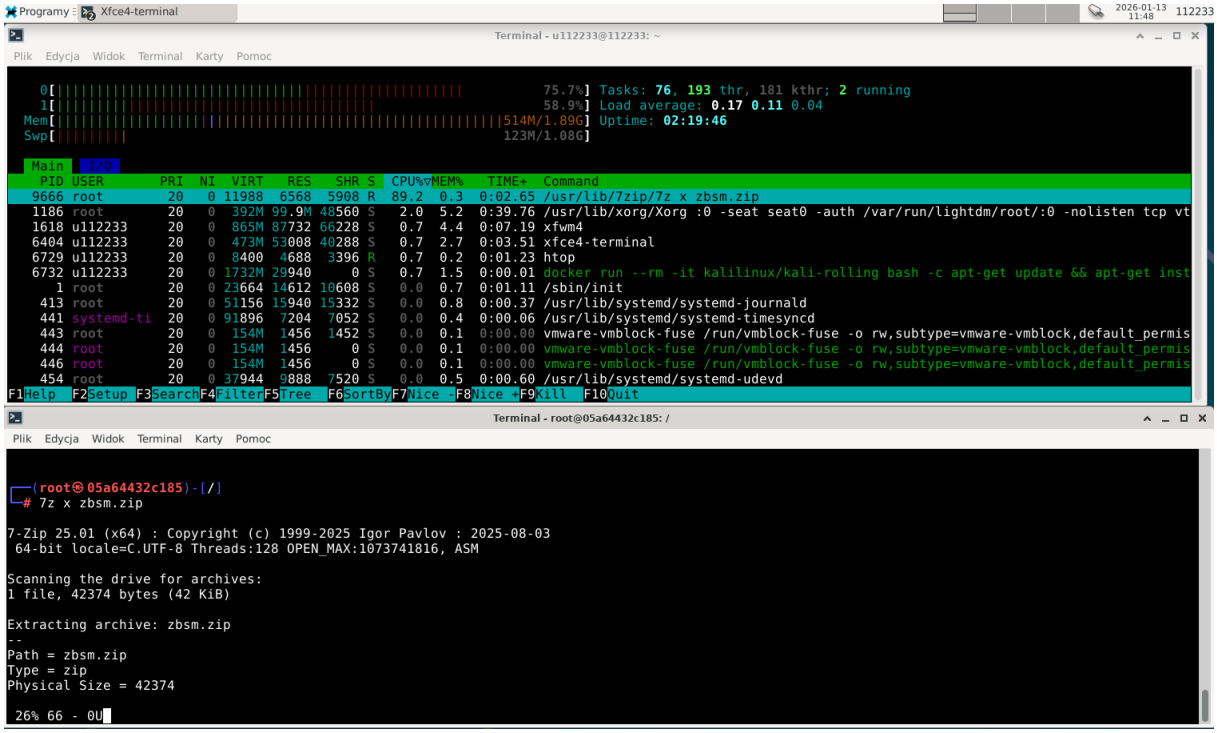
Uwaga: Zip bomba może całkowicie „zapchać” twój system operacyjny. W tym przypadku pobieramy 42 kB plik, który po wypakowaniu zajmuje 5 GB miejsca na dysku twardym! Istnieją również warianty, np. 10 MB, które po wypakowaniu zajmują 281 TB. **Zawsze działaj na maszynie wirtualnej!**

Uruchamianie zip bomby w sposób niekontrolowany jest nieodpowiedzialne i może być nielegalne, jeśli robisz to na cudzym sprzęcie lub w sieci!!!

Jeśli interesuje Cię temat zip bomb, możesz zapoznać się z artykułem Davida Fifielda, autorem jednej z najbardziej znanych i powszechnie cytowanych prac na temat tego typu ataku. Artykuł jest dostępny pod tym linkiem: https://www.bamsoftware.com/hacks/zipbomb/

8. Przyjrzyj się aktualnemu zachowaniu procesora, a następnie uruchom wypakowanie zip bomby: **(zadanie na 4.5)**

```
7z x zbsm.zip
```



Jak widzisz, obciążenie procesora znacznie wzrosło – na poszczególnych rdzeniach procesora często osiąga 100%. Oczywiście wartości te będą się różnić w zależności od posiadanego sprzętu. Aby zapobiec takim sytuacjom, musimy skonfigurować ograniczenia nakładane na kontener.

9. Jeżeli proces wypakowania trwa zbyt długo, przerwij go kombinacją klawiszy `Ctrl + C`.

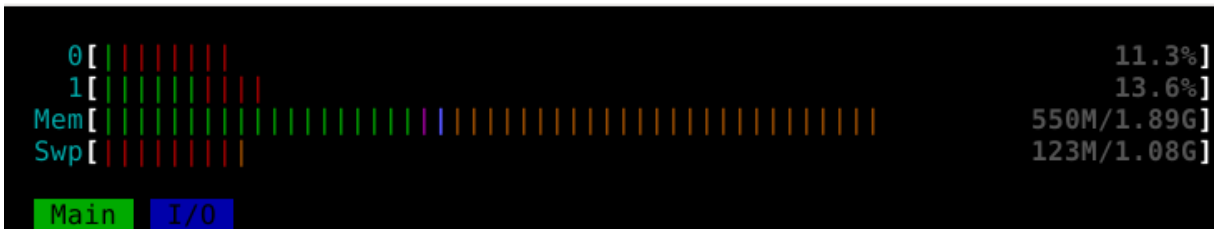
10. Zamknij kontener komendą `exit`.

11. Ponownie uruchom kontener, ale z dodatkowymi parametrami, które ograniczą dostęp do pamięci do poziomu 150 MB oraz do 20% wykorzystania procesora:

```
docker run --rm -it --memory="150m" --cpus="0.2" kalilinux/kali-rolling bash -c "apt-get update && apt-get install -y p7zip-full wget && bash"
```

12. Powtórz kroki 3.7 oraz 3.8.

13. Obserwuj zachowanie procesora. Jak zauważysz, obciążenie rdzeni oscyluje wokół 20%, co jest zgodne z oczekiwaniami. **(zadanie na 5.0)**



14. Jeżeli proces wypakowania trwa zbyt długo, przerwij go kombinacją klawiszy `Ctrl + C`, a następnie wyjdź z kontenera poleceniem `exit`.