



Krzysztof Godzisz

# Laboratorium cyberbezpieczeństwa w Dockerze

Zrób to sam

Helion 

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite/Olsztyn  
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: [helion@helion.pl](mailto:helion@helion.pl)

WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/twpila>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-8322-311-7

Copyright © Helion S.A. 2023

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

# Spis treści

---

<b>Przedmowy</b> .....	<b>7</b>
<b>Od autora</b> .....	<b>11</b>
<b>ROZDZIAŁ 1. Czym jest Docker</b> .....	<b>13</b>
1.1. Kontenery .....	13
1.2. Obrazy .....	14
1.3. Komunikacja między kontenerami .....	15
1.4. Czy Docker jest dla mnie? .....	16
Podsumowanie .....	16
<b>ROZDZIAŁ 2. Instalacja Dockera w systemach Linux</b> .....	<b>17</b>
2.1. Dystrybucja Debian .....	17
2.1.1. Instalacja z wykorzystaniem repozytorium Debiana .....	17
2.1.2. Instalacja z wykorzystaniem repozytorium Dockera .....	18
2.1.3. Instalacja określonej wersji dockera .....	19
2.2. Dystrybucja Ubuntu .....	20
2.2.1. Instalacja z wykorzystaniem repozytorium Ubuntu .....	20
2.2.2. Instalacja z wykorzystaniem repozytorium Dockera .....	21
2.2.3. Instalacja określonej wersji Dockera .....	22
2.3. Dystrybucja Fedora .....	23
2.3.1. Instalacja z wykorzystaniem repozytorium Fedory .....	23
2.3.2. Instalacja z wykorzystaniem repozytorium Dockera .....	23
2.3.3. Instalacja określonej wersji Dockera .....	24
2.4. Dystrybucja CentOS .....	25
2.4.1. Instalacja z wykorzystaniem repozytorium CentOS .....	25
2.4.2. Instalacja z wykorzystaniem repozytorium Dockera .....	26
2.4.3. Instalacja określonej wersji Dockera .....	27
2.5. Instalacja z paczki .....	28
2.5.1. Instalacja w systemach Debian oraz Ubuntu .....	28
2.5.2. Dla systemu Fedora .....	29
2.5.3. Dla systemu CentOS .....	30
2.6. Instalacja Dockera za pomocą plików binarnych .....	31
2.7. Sprawdzenie, czy instalacja zakończyła się powodzeniem .....	32
2.8. Błędy, z jakimi możesz się spotkać .....	32
2.8.1. Błąd podczas uruchomienia .....	32
2.8.2. Zduplikowane repozytoria .....	32
2.8.3. Brak uprawnień .....	33
2.9. Korzystanie z Dockera na koncie bez uprawnień administracyjnych .....	33
Podsumowanie .....	34

<b>ROZDZIAŁ 3. Obrazy Dockera .....</b>	<b>35</b>
3.1. Obrazy w Dockerze .....	35
3.1.1. Obrazy oficjalne .....	36
3.1.2. Obrazy użytkownika .....	37
3.2. Wyświetlanie obrazów .....	37
3.3. Usuwanie zbędnych obrazów .....	38
3.4. Obrazy dystrybucji Linux a obrazy systemu Windows .....	40
3.5. Jak szukać obrazów .....	41
Podsumowanie .....	42
<b>ROZDZIAŁ 4. Kontenery Dockera .....</b>	<b>43</b>
4.1. Tworzenie kontenera .....	43
4.2. Usuwanie zbędnych kontenerów .....	44
4.3. Uruchomienie kontenera .....	45
4.4. Uruchomienie istniejącego kontenera .....	47
4.5. Wejście do powłoki kontenera .....	48
4.6. Zatrzymanie działania kontenera .....	50
4.7. Dlaczego musimy tworzyć kontener w taki sposób? .....	50
Podsumowanie .....	51
<b>ROZDZIAŁ 5. Sieci Dockera .....</b>	<b>52</b>
5.1. Automatycznie generowana sieć Dockera .....	52
5.2. Własna sieć .....	54
5.3. Podłączenie do sieci .....	55
5.4. Odłączanie od sieci .....	55
5.5. Testujemy naszą sieć .....	56
5.6. Dodawanie kontenera do nowej sieci w trakcie jego tworzenia .....	57
Podsumowanie .....	57
<b>ROZDZIAŁ 6. Metasploitable2 i Docker .....</b>	<b>58</b>
6.1. Czym jest Metasploitable .....	58
6.2. Utworzenie własnej sieci .....	58
6.3. Pobranie obrazów oraz utworzenie kontenerów .....	59
6.4. Instalacja niezbędnego oprogramowania w kontenerze .....	60
6.5. Uzyskujemy dostęp do kontenera .....	60
Podsumowanie .....	62
<b>ROZDZIAŁ 7. Uruchamianie DVWA, czyli porty oraz kopiowanie i zamiana plików w kontenerze .....</b>	<b>63</b>
7.1. Pobranie obrazu DVWA .....	63
7.2. Utworzenie oraz uruchomienie kontenera .....	63
7.3. Operacje na plikach .....	65
7.3.1. Modyfikacja plików wewnątrz kontenera .....	65
7.3.2. Modyfikacja plików kontenera poza kontenerem .....	66
7.4. reCAPTCHA key .....	67
7.5. Start DVWA .....	67
Podsumowanie .....	67

<b>ROZDZIAŁ 8. Kontener Kali Linux z GUI .....</b>	<b>68</b>
8.1. Pobranie obrazu, uruchomienie kontenera i wszystko, co do tej pory poznałeś .....	68
8.2. Konfiguracja tightvncserver .....	69
8.3. Logowanie do kontenera .....	70
Podsumowanie .....	72
<b>ROZDZIAŁ 9. Dockerfile — przygotowujemy obraz Kali Linux .....</b>	<b>73</b>
9.1. Tworzenie pliku Dockerfile .....	73
9.2. Dodanie systemu bazowego .....	73
9.3. Określenie twórcy obrazu .....	74
9.4. Instalacja niezbędnego oprogramowania .....	74
9.5. Polecenia automatycznie uruchamiane w kontenerze .....	74
9.6. Utworzenie obrazu oraz jego uruchomienie .....	75
9.7. Serwer Apache .....	76
9.8. Opis etykiet, jakie możesz wykorzystać w Dockerfile .....	77
9.8.1. FROM .....	77
9.8.2. RUN .....	77
9.8.3. CMD .....	78
9.8.4. ENTRYPOINT .....	78
9.8.5. COPY i ADD .....	79
9.8.6. EXPOSE .....	79
9.8.7. ENV .....	79
9.8.8. VOLUME .....	79
9.8.9. USER .....	79
9.8.10. WORKDIR .....	80
Podsumowanie .....	80
<b>ROZDZIAŁ 10. Woluminy, czyli współdzielenie zasobów w Dockerze .....</b>	<b>81</b>
10.1. Tworzenie współdzielonej przestrzeni .....	81
10.2. Współdzielenie pomiędzy kontenerami .....	82
10.3. Miejsce montowania woluminów .....	83
10.4. Pliki dodawane do woluminu poza kontenerami .....	84
10.5. Wolumin udostępniony w określonym miejscu .....	84
10.6. Praktyczny przykład .....	85
10.7. Usuwanie woluminów .....	86
Podsumowanie .....	88
<b>ROZDZIAŁ 11. Multikontener — uruchamiamy Wordpressa .....</b>	<b>89</b>
11.1. Instalacja Docker Compose .....	89
11.2. Docker Compose i plik .yaml .....	90
11.3. Tworzymy pierwszy plik multikontenera .....	90
11.3.1. Wersja .....	90
11.3.2. Aplikacje — services .....	91
11.4. Uruchomienie multikontenera .....	94
11.5. Sieć .....	95
Podsumowanie .....	97

<b>ROZDZIAŁ 12. Instalacja oraz obsługa Greenbone Vulnerability Manager .....</b>	<b>98</b>
12.1. Instalacja Greenbone Vulnerability Manager .....	98
12.1.1. Tworzymy katalog .....	99
12.1.2. Dokładniejsze objaśnienie powyższego polecenia .....	99
12.1.3. Pobranie pliku .....	100
12.1.4. Pobranie obrazów oraz uruchomienie kontenerów .....	100
12.2. Zmiana hasła .....	102
12.3. Brak bazy SCAP .....	102
12.4. Dodawanie kontenerów do sieci .....	102
12.5. Uruchomienie Metasploitable2 .....	103
12.6. Konfigurowanie aplikacji do skanowania .....	104
12.6.1. Tworzenie celu .....	105
12.6.2. Przeprowadzenie skanowania .....	106
Podsumowanie .....	108
<b>ROZDZIAŁ 13. Greenbone Vulnerability Manager — raporty i wykorzystanie kilku luk .....</b>	<b>109</b>
13.1. Wyświetlanie raportu .....	109
13.2. Co to jest CVE oraz inne przydatne informacje .....	111
13.3. Wykorzystanie pierwszej luki .....	111
13.4. Samba MS-RPC Remote Shell Command Execution Vulnerability .....	112
13.5. DistCC RCE Vulnerability (CVE-2004-2687) .....	114
13.6. PostgreSQL weak password .....	116
Podsumowanie .....	117
<b>ROZDZIAŁ 14. Instalacja Juice Shop oraz przetestowanie kilku luk .....</b>	<b>118</b>
14.1. Pwning OWASP Juice Shop .....	118
14.1.1. Instalacja aplikacji .....	118
14.2. Więcej o Juice Shop .....	119
14.3. Pierwsze wyzwanie .....	120
14.4. Dostęp do konta administratora .....	121
14.5. XSS Injection .....	122
Podsumowanie .....	122
<b>Zakończenie. Co dalej? .....</b>	<b>123</b>
<b>DODATEK A Tworzenie konta na platformie DockerHub i przesyłanie do niej obrazów .....</b>	<b>125</b>
<b>DODATEK B Lista przydatnych narzędzi uruchamianych w Dockerze .....</b>	<b>132</b>
<b>DODATEK C Odnośniki do CTF uruchamianych w Dockerze .....</b>	<b>134</b>

## ROZDZIAŁ 4.

# Kontenery Dockera

---

Kontener w pierwotnej formie jest utworzonym z obrazu systemem bazowym. To dzięki kontenerowi uruchamiamy aplikację lub system w odizolowanym środowisku. Wykorzystując to rozwiązanie, możemy stworzyć laboratorium, które posłuży do nauki. Co ważniejsze, możemy wykonać wszystko w systemie, z którego korzystamy na co dzień, bez obawy, że coś w nim popsujemy.

To, co się dzieje wewnątrz kontenera, tam pozostaje. Działania przeprowadzane wewnątrz nie mają wpływu na właściwy system, dlatego możesz w ten sposób testować oprogramowanie typu *malware* bez ryzyka, że coś uszkodzisz.

## 4.1. Tworzenie kontenera

---

W poprzednim rozdziale pobraliśmy kilka obrazów. Po wprowadzonych zmianach nasza lista powinna wyglądać mniej więcej tak:

```
docker images
REPOSITORY          TAG          IMAGE ID      CREATED      SIZE
kalilinux/kali-rolling latest      cd70c686e299 7 days ago   126MB
ubuntu              latest      d2e4e1f51132 3 weeks ago  77.8MB
```

Mamy na nośniku systemy bazowe Kali Linux oraz Ubuntu. Zerknijmy teraz, w jaki sposób można utworzyć kontener z pierwszego z nich, czyli z `kalilinux/kali-rolling`:

```
docker create kalilinux/kali-rolling
11f5a76a04625ada47a2cd3299634df3a40b8035e39cc09d9146b97f3891ac3e
```

Utworzyliśmy kontener oraz zarezerwowaliśmy miejsce na niego i pliki przekopiowane z obrazu.

Nie otrzymaliśmy nic poza informacją, która nic nie wyjaśnia. Spójrzmy jednak na listę utworzonych kontenerów:

```
docker ps -a
CONTAINER ID  IMAGE                COMMAND          CREATED        STATUS          NAMES
11f5a76a0462  kalilinux/kali-rolling "bash"         5 minutes ago Created         vigilant_tesla
0e6886ec52df  feb5d9fea6a5       "/hello"       7 days ago    Exited (0) 7 days ago  distracted_yalow
```

Do wyświetlenia uruchomionych kontenerów służy polecenie `ps`. My ze względu na brak jakiegokolwiek uruchomionego kontenera skorzystaliśmy z opcji `-a`. Dzięki niej wyświetlane są wszystkie kontenery — te uruchomione i te tylko utworzone.

Obecnie do wyświetlania listy kontenerów zaleca się stosowanie polecenia `docker container ls`. Jest to nowsze rozwiązanie, które być może z czasem wyprze polecenie `ps`. Ja w książce będę korzystać z `ps`, ale jeśli chcesz, możesz używać nowszego.

Jeżeli wykonałeś wszystkie czynności z poprzedniego rozdziału, powinieneś mieć dwa kontenery. Jeden z nich utworzyłeś przed chwilą. Jest to `kalilinux/kali-rolling`. Drugi to kontener utworzony z usuniętego obrazu `hello-world`. Kontenerem `hello-world` zajmujemy się za chwilę, teraz utwórzmy z obrazu Kali Linux kolejny kontener. W tym celu wykonamy polecenie, które poznałeś przed chwilą:

```
docker create kalilinux/kali-rolling
da415c160f279718964730333dbe3d6eb465f072d80b40e08a83590a8cf1afb0
```

```
docker ps -a
CONTAINER ID   IMAGE                COMMAND             CREATED          STATUS          NAMES
da415c160f27   kalilinux/kali-rolling "bash"             3 seconds ago   Created         tender_beaver
11f5a76a0462   kalilinux/kali-rolling "bash"             25 minutes ago  Created         vigilant_tesla
0e6886ec52df   feb5d9fea6a5        "/hello"           7 days ago      Exited (0) 7 days ago  distracted_yalow
```

Dodatkowo utworzyliśmy nowy kontener, który różni się ID oraz nazwą wygenerowaną przez Dockera. Wszystko działa tak, jak powinno, niemniej taki zapis jest bardzo niewygodny dla oka oraz nie wyjaśnia, czym jest dany kontener. Najlepszy sposób to utworzyć kontener i nadać mu własną nazwę, dzięki której będzie można określić, do czego służy. Aby to zrobić, musisz skorzystać z poznanego wcześniej polecenia, do którego powinieneś dopisać opcję `--name`. Pokazuję to w poniższym przykładzie:

```
docker create --name system-kali kalilinux/kali-rolling
ae9e96e2d7cdf8c962508273ecb9d9a0312994ee63adb270d93a488e3d705d55

CONTAINER ID   IMAGE                COMMAND             CREATED          STATUS          NAMES
ae9e96e2d7cd   kalilinux/kali-rolling "bash"             4 seconds ago   Created         system-kali
```

Przykład skróciłem o nieistotne w danym momencie obrazy.

Jak na pewno zauważyłeś, na samym końcu, pod rubryką `names`, pojawiła się utworzona przez nas nazwa. Dzięki temu dokładniej wiemy, czym dany kontener jest i/lub do czego służy.

## 4.2. Usuwanie zbędnych kontenerów

Na każdym etapie operowania kontenerami powinniśmy dbać o porządek. Niestety każdy z nas wie, jak to czasami z tym bywa. Pomimo to przynajmniej od czasu do czasu wykonujemy czynności prowadzące do pozbycia się niepotrzebnych kontenerów. W Dockerze, jak i wszędzie indziej jest to bardzo istotne. Jeżeli mocno namnożymy kontenerów, to w pewnym momencie siłą rzeczy możemy mieć spore problemy z odnalezieniem tego, czego poszukujemy. Dlatego warto co jakiś czas pozbywać się niepotrzebnych bądź nieużywanych kontenerów.



Parę akapitów wcześniej wspominałem o zbędnym kontenerze `hello-world`. W ramach przypomnienia — jest to kontener:

```
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS          NAMES
0e6886ec52df   feb5d9fea6a5  "/hello"        7 days ago      Exited (0) 7 days ago  distracted_yalow
```

Oto sposób, w jaki możemy go usunąć tak, aby nie zajmował miejsca ani na ekranie, ani na nośniku:

```
docker rm 0e6886ec52df
0e6886ec52df
```

W powyższym przykładzie skorzystałem z ID danego kontenera. Możesz jeszcze posłużyć się nazwą nadaną w tym przypadku przez Dockera, czyli `distracted_yalow`. Jak sam zauważyłeś, nazwa ta nie mówi nam nic o tym, czym dany kontener jest lub co robi, dlatego warto nadawać nazwy samodzielnie.



Dla własnej wygody nadawaj swoim kontenerom odpowiednie nazwy.

Po usunięciu zbędnego kontenera nasza lista powinna wyglądać mniej więcej tak:

```
CONTAINER ID   IMAGE          COMMAND          CREATED          STATUS          PORTS          NAMES
ae9e96e2d7cd   kalilinux/kali-rolling  "bash"          2 hours ago      Created          system-kali
da415c160f27   kalilinux/kali-rolling  "bash"          2 hours ago      Created          tender_beaver
11f5a76a0462   kalilinux/kali-rolling  "bash"          2 hours ago      Created          vigilant_tesla
```

Mamy trzy kontenery utworzone z obrazu `kalilinux/kali-rolling`. Żadnego z nich już nie potrzebujemy, dlatego możemy wszystkie usunąć. Wystarczy, że skorzystamy z polecenia:

```
docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
ae9e96e2d7cdf8c962508273ecb9d9a0312994ee63adb270d93a488e3d705d55
da415c160f279718964730333dbe3d6eb465f072d80b40e08a83590a8c1afbf0
11f5a76a04625ada47a2cd3299634df3a40b8035e39cc09d9146b97f3891ac3e
Total reclaimed space: 0B
```



Polecenie `docker container prune` usunie wszystkie utworzone, ale nie uruchomione kontenery.

Przed wykonaniem polecenia Docker pyta, czy na pewno chcemy to zrobić. Rozpocznie usuwanie dopiero po wpisaniu `y` (ang. *yes*) i potwierdzeniu klawiszem *Enter*.

Aby potwierdzić usunięcie kontenerów, wystarczy posłużyć się przedstawionym wcześniej poleceniem `ps -a`.

## 4.3. Uruchomienie kontenera

W podrozdziale 2.7 korzystałeś z polecenia, aby sprawdzić, czy właściwie zainstalowałeś Dockera. Czas powiedzieć o tym więcej.

Przed chwilą poznałeś sposób na tworzenie kontenerów z obrazów. Wykorzystywałeś do tego polecenie `create`. Po utworzeniu takiego kontenera nic się nie działo — tylko się pojawił na liście. Sposób zaprezentowany wcześniej nie jest jedynym i przyznam, że większość użytkowników Dockera częściej stosuje ten, który poznasz za chwilę. Oczywiście mówimy o poleceniu `run`, które możemy nazwać poleceniem *dwa w jednym*. Otóż automatycznie tworzy ono kontener z obrazu, po czym go uruchamia. Zobaczmy to na przykładzie:

#### **docker run hello-world**

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:80f31da1ac7b312ba29d65080fddf797dd76acfb870e677f390d5acba9741b17
Status: Downloaded newer image for hello-world:latest
```

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:  
\$ `docker run -it ubuntu bash`

Share images, automate workflows, and more with a free Docker ID:  
<https://hub.docker.com/>

For more examples and ideas, visit:  
<https://docs.docker.com/get-started/>

Na samym początku możemy wyczytać, że obraz zostaje pobrany:

```
latest: Pulling from library/hello-world
2db29710123e: Pull complete
```

Następnie otrzymujemy informację od kontenera:

```
Hello from Docker!
```

Oznacza ona, że kontener został uruchomiony i wykonał wszystkie czynności, do jakich go skonfigurowano, a następnie zakończył pracę. Potwierdźmy informację, że kontener został utworzony:

```
docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          NAMES
2aefc4280725  hello-world   "/hello"                 6 minutes ago   Exited (0) 6 minutes ago   festive_archimedes
```

Zgodnie z otrzymanym wynikiem polecenie potwierdziło utworzenie kontenera, ale nie jest on uruchomiony. Lista uruchomionych kontenerów jest pusta (`docker ps`). Polecenie to wprowadza jednak pewne zamieszanie, szczególnie wśród osób rozpoczynających naukę Dockera. Nim jednak wyjaśnię, o co chodzi, wpiszmy `run` w taki sam sposób jak kilka przykładów wcześniej. Następnie wyświetlimy listę utworzonych kontenerów:

```
docker run hello-world
```

```
Hello from Docker!
```

```
...
```

```
docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
7c0150482458	hello-world	"/hello"	4 seconds ago	Exited (0) 4 seconds ago	pedantic_poitras
2aefc4280725	hello-world	"/hello"	14 minutes ago	Exited (0) 14 minutes ago	festive_archimedes

Jak widać na powyższym przykładzie, utworzyliśmy drugi kontener z tego samego obrazu.



Bardzo ważne jest to, że polecenie run **zawsze tworzy nowy kontener i go uruchamia**. Jednak **nie służy do uruchomienia już istniejącego kontenera**.



Używając polecenia run, możesz skorzystać z opcji nadawania własnej nazwy. W razie wątpliwości zerknij na polecenie poniżej:

```
docker run --name kontener hello-world
```

To, co nastąpi po wprowadzeniu powyższego polecenia, pokazałem już w paru miejscach. Dlatego teraz pozwolę sobie pominąć tę część, jak również część z wypisywaniem utworzonych kontenerów. Nim przejdziesz dalej, warto, żebyś sam poćwiczył użycie poznanych poleceń.

## 4.4. Uruchomienie istniejącego kontenera

Gdyby jedynym sposobem na korzystanie z kontenerów było to, co prezentowałem w poprzednim podrozdziale, użycie Dockera nie miałyby sensu. Dlatego każdy kontener można uruchamiać dowolną ilość razy. Dla przykładu, aby ponownie uruchomić kontener, wystarczy wpisać polecenie:

```
docker start -i kontener
```

Pozwoli ono wyświetlić informację startową kontenera hello-world. Jeślibyśmy nie wpisali opcji -i, wyświetliłaby się tylko nazwa kontenera, natomiast w wyniku jej użycia uruchamiamy standardowy strumień wyjścia. Inaczej pisząc, zostanie uruchomiony proces, który wyświetli informację ustawioną w kontenerze.

### Po czym można poznać, co zostanie wykonane po uruchomieniu kontenera?

Otóż wyświetlmy ostatni kontener obrazu hello-world, jaki utworzyliśmy:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
39c71a39ac8e	hello-world	"/hello"	4 hours ago	Exited (0) 20 minutes ago		kontener

W kolumnie COMMAND masz zapisane, jakie polecenie zostanie wykonane po uruchomieniu. W większości kontenerów można to zmienić. Lecz nim do tego przejdziemy, utworzymy sobie kontener na podstawie obrazu kalilinux/kali-rolling.

```
docker create kalilinux/kali-rolling
```

```
a7439791f06ddb188c2a2ea259b06f80270c440f034858d0eecf4aad8ab6cca7
```

Powinniśmy otrzymać taki oto kontener:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a7439791f06d	kalilinux/kali-rolling	"bash"	4 seconds ago	Created		pedantic_volhard

Jak widać w kolumnie `COMMAND`, kontener ma wpisane polecenie `bash`. Do tego jeszcze wrócimy. Teraz chciałbym pokazać, że możemy zmienić polecenie, które zostanie automatycznie uruchomione z kontenerem. Zrobimy to w następujący sposób:

```
docker create kalilinux/kali-rolling echo "Uczę się dockera"
847df1af4a9bd8f04d13f1eab20e1c21bce6f8045e8788ff7aca63d5d1e5abfd
```

Otrzymamy kontener:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
847df1af4a9b	kalilinux/kali-rolling	"echo 'Uczę się dock...'"	4 seconds ago	Created	wonderful_leakey

Możemy w ten sposób zmieniać polecenia po uruchomieniu kontenera. Teraz, aby sprawdzić, czy to zadziała, uruchamiamy nasz kontener:

```
docker start -i 847df1af4a9b
Uczę się dockera
```

## 4.5. Wejście do powłoki kontenera

---

Wiesz już, że możesz decydować o tym, co zostanie uruchomione w kontenerze, a teraz wejdziemy do środka. Do większości kontenerów można zalogować się tak samo jak do systemu Linux. Robi się to najczęściej po to, by sprawdzić, dlaczego dana aplikacja nie działa, lub aby doinstalować jakieś niezbędne oprogramowanie. Choć w tym drugim zastosowaniu najczęściej posługujemy się innym sposobem, ten poznasz już niedługo.

Aby dostać się do wiersza poleceń kontenera, musi on być utworzony z odpowiednimi opcjami uruchomienia. Nim przejdziemy do opisu, chciałbym, abyś utworzył jeden kontener z obrazu Kali Linux w sposób poznany wcześniej i jeden tak, jak zaraz pokażę. W ramach powtórzenia utworzę zarówno jeden, jak i drugi.

Tak, aby utrzymać porządek, pozbywam się wszystkich utworzonych do tej pory kontenerów:

```
docker container prune
```

Następnie tworzę kontener z obrazu Kali Linux w sposób, który poznałeś we wcześniejszym podrozdziale:

```
docker create --name kali-stary-sposob kalilinux/kali-rolling
```

Teraz tworzymy kontener w sposób umożliwiający zalogowanie się:

```
docker create --tty --interactive --name kali-cli kalilinux/kali-rolling
```

Wyjaśnijmy teraz, co zrobiliśmy.

Jako pierwsza pojawiła się opcja `--tty`. Pozwala ona uruchomić powłokę, w tym przypadku `bash`, w kontenerze Dockera. Druga, `--interactive`, utrzymuje otwarty standardowy strumień wejścia, dzięki któremu jesteśmy w stanie w `cli` kontenera wydawać polecenia.

Mamy dwa kontenery utworzone na podstawie tego samego obrazu. Jeden utworzyliśmy w sposób, który poznałeś wcześniej, a drugi — w nowy. Polecenie, którym posłużyliśmy się do utworzenia obydwu, jedynie tworzy kontener, a nie uruchamia. Dlatego aby móc się zalogować, musimy go uruchomić. Robimy to poleceniem:

```
docker start kali-cli
```

Jako odpowiedź otrzymasz nazwę swojego kontenera. Zamiast utworzonej nazwy możesz posłużyć się ID kontenera. Ja w tym przypadku zawsze staram się nazywać kontenery w zrozumiały dla mnie sposób, dlatego głównie korzystam z ich nazw.

Po uruchomieniu kontener powinien się pojawić na liście działających. Sprawdźmy to, tak jak w poniższym przykładzie:

```
docker ps
CONTAINER ID   IMAGE                COMMAND             CREATED          STATUS              PORTS              NAMES
84494c563982  kalilinux/kali-rolling  "bash"             22 minutes ago  Up 3 minutes              kali-cli
```

Aby można było użyć polecenia, które umożliwi korzystanie z `cli` kontenera, musi być on uruchomiony. Wykonaliśmy te czynności powyżej. Zobaczmy teraz, jak możemy się do niego zalogować:

```
docker exec -it kali-cli /bin/bash
```

Po wprowadzeniu polecenia zostaniemy zalogowani do kontenera i pojawi się powłoka `bash` systemu Kali Linux. Wszystkie polecenia są dostępne, natomiast o instalację oprogramowania musimy zadbać sami.

Powyższe polecenie daje możliwość wykonywania wszelkich znanych nam operacji związanych z terminalem.

Opcje `-it` to ogólny skrót oraz połączenie dwóch przedstawionych wcześniej, czyli `--interactive` oraz `--tty`. Aby sprawdzić, czy wszystko działa, wprowadź kilka znanych Ci poleceń, a zobaczysz wyniki identyczne z tymi, jakie zobaczyłeś w systemie Linux.

Aby wyjść z kontenera, możesz wpisać `exit` i wcisnąć *Enter* lub skorzystać z kombinacji klawiszy *Ctrl+D*.

Wszystko, co wykonasz w danym kontenerze, zostanie zapisane. Jeżeli chcesz to sprawdzić, utwórz katalog lub plik o dowolnej nazwie, a następnie wyjdź z kontenera i zaloguj się do niego ponownie.

Jak już wiesz, opcje `-it` są skrótową wersją tych, które wcześniej poznałeś. Jeżeli chcesz, w każdym przypadku, niezależnie od polecenia, możesz zamiast wersji opisowej stosować tę skróconą. Ja specjalnie na początku wybrałem te bardziej opisowe, by lepiej było widoczne to, co robię, jednak teraz zdajesz sobie sprawę, do czego służą, więc możesz śmiało korzystać z wersji skróconej. W każdym razie niezależnie od Twojego wyboru wiedź, że w dalszej części książki będę się posługiwał wersją skróconą.

## 4.6. Zatrzymanie działania kontenera

---

Pomimo tego, że opuściliśmy kontener, nadal jest uruchomiony. Potwierdza to wynik wykonania polecenia:

```
docker ps
CONTAINER ID   IMAGE                COMMAND             CREATED          STATUS              PORTS              NAMES
84494c563982  kali/linux/kali-rolling "bash"             41 minutes ago  Up 22 minutes      kali-cli
```

Samo wyjście z kontenera nie kończy jego działania. Aby zakończyć działanie kontenera, posługujemy się bardzo prostym poleceniem:

```
docker stop kali-cli
```

Po odczekaniu chwili i ponownym uruchomieniu polecenia ps kontener nie będzie widoczny na liście, czyli nie będzie uruchomiony.

Jednak pamiętaj: zatrzymanie działania kontenera nie oznacza, że zmiany, jakie w nim wprowadziliśmy, zostały utracone. One nadal tam są, tylko aby się do nich dostać, musimy ponownie uruchomić kontener i się załogować.

## 4.7. Dlaczego musimy tworzyć kontener w taki sposób?

---

Na koniec chciałbym zająć Ci jeszcze chwilę i wspomnieć o sposobie działania Dockera.

Nie raz wspominałem, że kontenery Dockera powstają z obrazów. To my decydujemy, jak nasz kontener będzie działał. Czyli to my decydujemy o tym, w jaki sposób my, jak i inni użytkownicy będziemy z niego korzystać. Abyś to lepiej zrozumiał, spróbujmy zrobić to samo co wyżej z kontenerem utworzonym w sposób, który poznałeś na samym początku tego rozdziału:

```
docker start kali-stary-sposob
kali-stary-sposob
```

Otrzymałeś taką samą odpowiedź jak w przypadku utworzenia kontenera z opcjami `-it`, ale po uruchomieniu polecenia ps kontener nie pojawił się na liście. Powodem jest to, że zostały uruchomione wszystkie polecenia, jakie zakładała podstawowa konfiguracja obrazu. Następnie kontener został niezauważalnie zamknięty.

Natomiast po użyciu opcji `-it` kontener uruchomił się i działa, dopóki my go nie zamknijemy. Nie chcę, abyś pomyślał, że automatyczne zamykanie kontenera to coś złego, ponieważ czasami taka opcja jest potrzebna. *Wykonać to, co ma wykonać, i się wyłączyć.* Natomiast my z Kali Linux chcemy korzystać dość intensywnie, dlatego dostęp do powłoki jest niezbędny.

Nie mamy uruchomionego kontenera, dlatego nie możemy się do niego zalogować. Potwierdźmy to jednak poleceniem:

```
docker exec -it kali-stary-sposob /bin/bash
```

```
Error response from daemon: Container db14561c34973ca161d2d09925d4a0fcc20633ea1e51f439403eb43a28b7112a  
is not running
```

Otrzymaliśmy informację o tym, że kontener nie jest uruchomiony, więc nie mamy dostępu do powłoki.

## Podsumowanie

---

Wiesz już, jak tworzyć kontenery, zarządzać nimi, uruchamiać powłoki. Nie jest to jednak wszystko, co powinieneś wiedzieć. Do samych kontenerów jeszcze wrócimy, ponieważ jest to bardziej rozbudowane zagadnienie, niż tutaj opisałem. Na tym etapie chciałbym, żebyś zatrzymał się i przeanalizował nabytą wiedzę. Utworzył kilka własnych kontenerów, poszukał innych obrazów i z nimi zrobił rzeczy podobne do tych, które robiliśmy w tym rozdziale. Wierz mi, na pewno się opłaci. Gdy poczujesz się już pewniej, przejdź do następnego rozdziału. W nim porozmawiamy o sieciach w Dockerze.





# PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA  
**Helion** 

**Izolacja procesów i środowisk** to dziś jeden z najważniejszych mechanizmów zapewniania bezpieczeństwa systemów IT. Dzięki temu możemy nie tylko bezpiecznie uruchamiać niezauwane zasoby, ale przede wszystkim przeprowadzać testy penetracyjne i badać złośliwe oprogramowanie bez ryzyka, że zainfekuje ono natywne środowisko. Jedną z najwyższych form izolacji jest konteneryzacja – użytkownik ma wówczas praktycznie całkowitą kontrolę nad relacją pomiędzy kontenerem a systemem operacyjnym hosta.

W ostatnich latach na lidera konteneryzacji wyrosło środowisko uruchomieniowe Docker. Dzięki funkcjonalności, wygodnej obsłudze i szerokiej kompatybilności z różnymi systemami operacyjnymi stanowi on dziś techniczny standard. Docker otwiera przed badaczami cyberbezpieczeństwa nowe możliwości testowania oprogramowania pod kątem bezpieczeństwa i analizy wszelkiej maści malware bez konieczności budowania kosztownej infrastruktury – dzięki Dockerowi i konteneryzacji swoje laboratorium można zbudować w obrębie pojedynczej stacji roboczej.

*Laboratorium cyberbezpieczeństwa w Dockerze. Zrób to sam* to przewodnik instruujący krok po kroku, jak zbudować własne laboratorium w kontenerze Dockera. Szczegółowo omawia proces instalacji środowiska na różnych dystrybucjach Linuksa i jego konfiguracji, a także kwestie związane z zarządzaniem kontenerami i budowaniem zależności.

## Najważniejsze zagadnienia:

- instalacja i konfiguracja Dockera
- zarządzanie kontenerami
- budowanie sieci Dockera
- obsługa narzędzi pentesterskich
- budowanie i uruchamianie multikontenerów
- zarządzanie podatnościami



**Zbuduj własny cybersec lab –  
najlepiej w Dockerze!**

	<b>KOD KORZYŚCI</b> Sięgnij po więcej! ▶		Patron medialny:
<b>helion.pl</b>			
<b>HELION SA</b> ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	ISBN 978-83-8322-311-7		
Cena: 49,00 zł			