Jeeva S. Chelladhurai,
Vinod Singh, Pethuru Raj

# Docker dla praktyków

## Wydanie II

Helion

Packt›

Tytuł oryginału: Learning Docker - Second Edition

Tłumaczenie: Konrad Matuk

ISBN: 978-83-283-3972-9



Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
*http://helion.pl/user/opinie/docpr2*
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

# Spis treści

ROZDZIAŁ 2

# Obsługa kontenerów Dockera

W poprzednim rozdziale wyjaśniliśmy, dlaczego Docker to narzędzie przyszłości umożliwiające tworzenie uniwersalnych kontenerów dostosowanych do wymogów poszczególnych aplikacji. Dowiedziałeś się, że Docker pozwala na przenoszenie kontenerów pomiędzy środowiskami (lokalnymi i zdalnymi). Zapewne widzisz już potencjalne zastosowania Dockera w środowisku, w którym pracujesz. Teraz czas, abyś zrozumiał zagadnienia związane z cyklem roboczym kontenera. W tym rozdziale dowiesz się, jak optymalnie korzystać z kontenerów własnych i tych, które zostały opracowane przez innych, aby uzyskać jak najlepszą wydajność i uniknąć potencjalnych problemów. Kontenery mogą usprawnić pracę nad aplikacją, a także jej testowanie i dystrybucję.

Podczas lektury tego rozdziału poznasz najważniejsze zagadnienia związane z obsługą kontenerów. Udzielimy Ci wielu praktycznych porad i przedstawimy polecenia, które pozwolą Ci usprawnić pracę z kontenerami.

W rozdziale zajmiemy się następującymi tematami:

- wyjaśnienie terminów związanych z Dockerem;
- praca z obrazami i kontenerami Dockera;
- funkcje serwisu Docker Registry i jego repozytorium;
- usługa Docker Hub Registry;
- szukanie obrazów Dockera;
- praca z interaktywnymi kontenerami;
- śledzenie zmian wewnątrz kontenera;
- kontrolowanie i utrzymywanie kontenerów Dockera;
- tworzenie obrazów na podstawie kontenerów;
- uruchamianie kontenera jako demona.

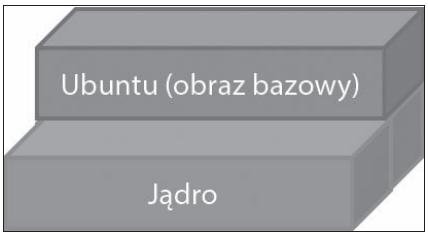
# Wyjaśnienie terminów związanych z Dockerem

W kolejnych sekcjach wyjaśnimy znaczenie częściej używanych terminów związanych z Dockerem. Dzięki temu podrozdziałowi łatwiej będzie Ci zrozumieć zagadnienia poruszane w dalszej części tego rozdziału.

## Obrazy Dockera

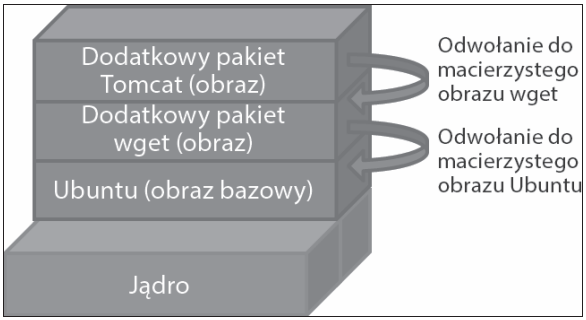
**Obraz Dockera** jest zbiorem wszystkich plików tworzących wykonywalną aplikację. Zawiera on aplikację oraz wszystkie biblioteki, pliki binarne i inne elementy zależne, takie jak np. deskryptor wdrożenia, które umożliwiają uruchomienie aplikacji w dowolnym środowisku. Pliki znajdujące się w obrazie Dockera są przeznaczone tylko do odczytu, więc nie można edytować zawartości obrazu. Wprowadzanie zmian zawartości obrazu jest możliwe tylko po dodaniu do niego kolejnej warstwy — obrazy Dockera składają się z warstw, na podstawie których można odtworzyć historię prac. Służy do tego polecenie `docker history` (więcej informacji na ten temat znajdziesz w rozdziale 3., „Budowanie obrazów").

Architektura obrazów Dockera skutecznie rozbudowuje koncepcję warstw, umożliwiając dodawanie nowych funkcjonalności do istniejących obrazów w celu dostosowania ich do zmieniających się wymagań biznesowych i wielokrotne używanie tych samych obrazów. Innymi słowy: dodawanie nowych warstw umożliwia dodawanie nowych funkcjonalności do stworzonych wcześniej obrazów, co pozwala tworzyć na ich podstawie nowe obrazy. Obrazy Dockera charakteryzują się hierarchicznymi zależnościami. Obraz znajdujący się najniżej w strukturze zależności określa się mianem **obrazu bazowego**. Obraz bazowy nie ma żadnego elementu macierzystego:




Na powyższym schemacie *Ubuntu* jest obrazem bazowym — taki obraz nie ma elementu macierzystego.

> Ubuntu jest systemem operacyjnym Linuksa opartym na systemie Debian. Obraz *Ubuntu* Dockera jest minimalistycznym zbiorem bibliotek programu i plików binarnych wymaganych do uruchomienia aplikacji. Obraz ten nie zawiera jądra systemu Linux, sterowników zapewniających obsługę urządzeń ani różnych usług wchodzących w skład pełnoprawnego systemu Ubuntu.



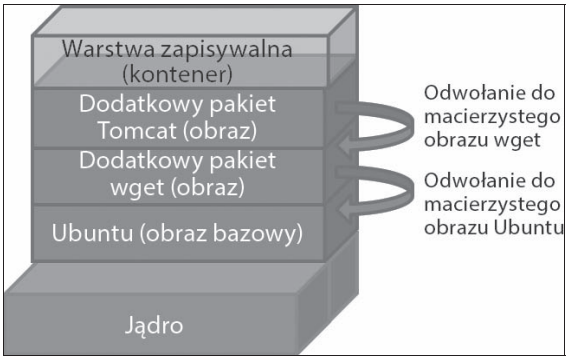

Jak widzisz, wszystko zaczyna się od obrazu bazowego. Obrazem takim na zaprezentowanym rysunku jest *Ubuntu*. Możliwości tego obrazu rozszerza dodatkowa warstwa — obraz *wget*. Obraz *wget* odwołuje się do obrazu *Ubuntu* jak do obrazu nadrzędnego. Kolejna warstwa dodaje aplikację serwerową *Tomcat*, dla której obraz *wget* jest obrazem nadrzędnym. Każdy element dodawany do oryginalnego obrazu bazowego jest przechowywany w oddzielnej warstwie (hierarchia organizacji obrazu pozwala na określenie jego tożsamości). Każdy obraz Dockera powstaje na podstawie obrazu bazowego, do którego dodawane są kolejne funkcje w postaci nowych modułów będących kolejnymi warstwami opartymi na istniejącym obrazie Dockera (proces ten zilustrowano na powyższych rysunkach).

Platforma Docker umożliwia łatwe tworzenie nowych obrazów i rozbudowywanie już istniejących. Obrazy Dockera opracowane przez innych użytkowników tej platformy możesz pobierać z prywatnych i publicznych repozytoriów. Każdy obraz ma unikalny identyfikator (więcej informacji na ten temat znajdziesz w kolejnej sekcji).

## Kontenery Dockera

Obrazy Dockera to tak naprawdę szablony przeznaczone tylko do odczytu. Zawierają one wszystkie składniki niezbędne do uruchomienia aplikacji, ale nie przechowują żadnych informacji o jej stanie. Kontener Dockera jest tworzony na podstawie obrazu Dockera. Zawiera dodatkową warstwę przeznaczoną do odczytu i zapisu, która znajduje się nad statycznymi warstwami obrazu. Gdybyśmy chcieli zastosować porównania do terminów związanych z programowaniem obiektowym, to obrazy Dockera są klasami, a kontenery Dockera są obiektami (instancjami klas).

Obraz Dockera definiuje zachowanie kontenera Dockera — określa proces włączany podczas uruchamiania kontenera. W poprzednim rozdziale korzystałeś z polecenia `docker run hello-world`, które sprawiało, że silnik Dockera uruchamiał nowy kontener utworzony na podstawie obrazu *hello-world*, co powodowało wyświetlenie na ekranie komputera dość dużej ilości informacji. To chyba oczywiste, że obrazy Dockera są podstawowym blokiem konstrukcyjnym kontenerów Dockera. Obrazy określają sposób pracy kontenerów.




Na powyższym rysunku przedstawiono warstwę zapisywalną (umożliwiającą zapis i odczyt danych) dodawaną do obrazu podczas tworzenia kontenera w celu zapisu stanu aplikacji. Pod zapisywalną warstwą kontenera może się znajdować kilka obrazów przeznaczonych tylko do odczytu.

## Rejestr Dockera

**Rejestr Dockera** to repozytorium przeznaczone do przechowywania prywatnych i publicznych obrazów Dockera. Umożliwia szybkie tworzenie aplikacji przez programistów znajdujących się w dowolnym miejscu. Wszystkie obrazy przechowywane w rejestrze przechodzą proces wielokrotnej walidacji, weryfikacji i modyfikacji, a więc ich jakość jest naprawdę wysoka. Możesz załadować swój obraz Dockera do rejestru za pomocą polecenia `docker push`. Polecenie `docker pull` służy do pobierania obrazów z rejestru.

Rejestr Dockera może być prowadzony przez dowolnego usługodawcę i może mieć charakter publiczny lub prywatny. Oto przykładowe rejestry:

- Docker Hub,
- Quay,
- Google Container Registry,
- AWS Container Registry.

Każda firma i każdy programista może utworzyć własny rejestr przeznaczony tylko do przechowywania lub także udostępniania obrazów Dockera.

# Praca z obrazami Dockera

W poprzednim rozdziale pokazaliśmy typowy przykład wyświetlający komunikat w rodzaju `Witaj, świecie!` — uruchomiliśmy obraz *hello-world*. Czas dokładniej przeanalizować dane wyświetlane po uruchomieniu polecenia `docker pull` używanego do pobierania obrazów Dockera.

W dalszej części tego podrozdziału będziemy korzystać z obrazu *busybox* — jednego z najmniejszych, ale bardzo praktycznych obrazów Dockera. Przykład ten pozwoli nam zrozumieć sposób obsługi obrazów przez Dockera:

```
$ sudo docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
8ddc19f16526: Pull complete
Digest: sha256:a59906e33509d14c036c8678d687bd4eec81ed7c4b8ce907b888c607f6a1e0e6
Status: Downloaded newer image for busybox:latest
```

Przyjrzyj się komunikatom wyświetlonym po uruchomieniu polecenia `docker pull`. Jednym z nich jest *Using default tag: latest* (używanie domyślnej etykiety: najnowszy). Rejestry obrazów Dockera obsługują mechanizmy zarządzania, które pozwalają na przechowywanie wielu wariantów tego samego obrazu — różne wersje obrazu mogą być oznaczone określonymi etykietami.

Domyślnie Docker zawsze pobiera obraz oznaczony etykietą *latest* (najnowszy). Każdy wariant obrazu może być wybrany za pomocą etykiety. Etykietę należy umieścić po dwukropku (`:`) wpisanym po nazwie repozytorium (`<repozytorium>:<etykieta>`). Spróbujmy pobrać obraz *busybox* oznaczony etykietą *1.24*:

```
$ sudo docker pull busybox:1.24
1.24: Pulling from library/busybox

385e281300cc: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:8ea3273d79b47a8b6d018be398c17590a4b5ec604515f416c5b797db9dde3ad8
Status: Downloaded newer image for busybox:1.24
```

To doskonała funkcja, prawda? Możemy pobrać dowolną wersję obrazu *busybox*. W zaprezentowanym przykładzie pobraliśmy wersję 1.24. Polecenie `docker pull` obsługuje również parametr `-a`, który służy do pobierania wszystkich dostępnych wersji obrazu. Zachowaj ostrożność, korzystając z niego, bo możesz bardzo szybko zapełnić całą dostępną przestrzeń dyskową.

Pobraliśmy już kilka obrazów Dockera ze zdalnego repozytorium. Teraz host Dockera może uzyskać lokalny dostęp do pobranych obrazów. W celu wyświetlenia listy obrazów dostępnych dla hosta Dockera skorzystaj z polecenia `docker images`:

```
$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
hello-world         latest              c54a2cc56cbb        3 weeks ago
1.848 kB
busybox             latest              2b8fd9751c4c        4 weeks ago
1.093 MB
busybox             1.24                47bcc53f74dc        4 months ago
1.113 MB
```

Na wyświetlonej liście znalazły się trzy elementy. Przyjrzyjmy się bliżej danym wygenerowanym po uruchomieniu polecenia `docker images`. Oto lista kolumn, które mogą być wyświetlone przez to polecenie:

- `REPOSITORY` (repozytorium) — nazwa repozytorium lub obrazu. W poprzednich przykładach korzystaliśmy z repozytoriów o nazwach *hello-world* i *busybox*.
- `TAG` (etykieta) — etykieta przypisana do obrazu, np. *1.24* lub *latest*. Do obrazu może być przypisanych kilka etykiet.
- `IMAGE ID` (identyfikator obrazu) — do każdego obrazu przypisany jest unikalny identyfikator. Jest to losowa liczba szesnastkowa mająca 64 cyfry. Domyślnie polecenie `docker images` wyświetla tylko 12 cyfr. W celu wyświetlenia wszystkich 64 cyfr skorzystaj z flagi `--no-trunc` (np. `sudo docker images --no-trunc`).
- `CREATED` (utworzony) — czas, który upłynął od utworzenia obrazu.
- `SIZE` (rozmiar) — wirtualny rozmiar obrazu.

## Serwis Docker Hub

W poprzedniej sekcji uruchamiałeś polecenie `docker pull` w celu pobrania obrazu *busybox*. W tej sekcji chciałbym opisać działanie tego polecenia. Dowiesz się, że polecenie `docker pull` korzysta z serwisu Docker Hub.

Społeczność rozwijająca Dockera doprowadziła do powstania publicznego repozytorium obrazów dostępnego pod domyślnym adresem *index.docker.io*. Ta domyślna lokalizacja to właśnie **Docker Hub**. Polecenie `docker pull` jest zaprogramowane tak, aby szukało obrazów pod tym właśnie adresem. W związku z tym jeżeli uruchomisz polecenie pobierania obrazu *busybox*, to obraz ten zostanie pobrany z domyślnego rejestru. Taki mechanizm przyśpiesza uruchamianie kontenerów Dockera. Docker Hub to oficjalne repozytorium zawierające wszystkie obrazy utworzone i opublikowane przez społeczność rozwijającą Dockera. W serwisie zaimplementowano rozwiązania polegające na tym, że obrazy przechodzą procedury kontrolne. Ponadto serwis weryfikuje tożsamość osoby publikującej obraz, a także integralność wszystkich danych przepływających przez wszystkie kanały rejestru.

Istnieją sprawdzone metody weryfikacji i walidacji pozwalające na usunięcie z obrazów Dockera wszystkich celowo lub przypadkowo wprowadzonych wirusów i złośliwego oprogramowania. Integralność obrazów Dockera jest zapewniana przez mechanizm cyfrowego podpisu. Jeżeli mimo to oficjalny obraz zostanie uszkodzony lub sfałszowany, to silnik Dockera wyświetli informujące o tym ostrzeżenie przed uruchomieniem obrazu.

Poza oficjalnym repozytorium serwis Docker Hub Registry udostępnia platformę dla niezależnych programistów, która może być użyta do upublicznienia opracowanych przez nich obrazów. Obrazy opracowane przez niezależnych programistów są oznaczone identyfikatorem użytkownika (identyfikator ten jest umieszczany przed nazwą obrazu). Na przykład obraz *thedockerbook/helloworld* został opracowany przez niezależnego programistę o identyfikatorze *thedockerbook* i jest udostępniany w repozytorium o nazwie *helloworld*. Obrazy opracowane przez niezależnych twórców można pobierać za pomocą polecenia `docker pull`:

```
$ sudo docker pull thedockerbook/helloworld
```

Poza pobieraniem obrazów z oficjalnego repozytorium Docker umożliwia pobieranie obrazów z repozytoriów niezależnych (mogą być to repozytoria publiczne lub prywatne). Jak zapewne pamiętasz, silnik Dockera domyślnie szuka obrazów pod adresem *index.docker.io*. Jeżeli chcesz pobrać obraz z innego repozytorium, to musisz ręcznie określić ścieżkę, pod którą znajduje się dany obraz. Adres obrazu przypomina adres URL, ale nie zawiera specyfikatora protokołu (np.: *https://*, *http://* lub *ftp://*). Oto przykładowe polecenie pobierające obraz z niezależnego repozytorium:

```
$ sudo docker pull rejestr.przykład.com/moja_aplikacja
```

## Wyszukiwanie obrazów Dockera

Repozytorium Docker Hub udostępnia oficjalne obrazy, a także obrazy opracowane przez niezależnych programistów — użytkowników Dockera. W chwili pisania tej książki Docker Hub udostępnia tysiące obrazów (określanych mianem **zdockeryzowanych aplikacji**). Większość z nich została pobrana przez miliony użytkowników. Obrazy te mogą być używane w sposób bezpośredni lub pełnić funkcję bloku konstrukcyjnego aplikacji tworzonych przez użytkownika Dockera.

Zawartość oficjalnego repozytorium możesz przeszukiwać za pomocą polecenia `docker search`:

```
$ sudo docker search mysql
```

W wynikach wyszukiwania frazy *mysql* znajdziesz wiele obrazów związanych z technologią MySQL. Ograniczmy wyświetlaną listę do kilku pierwszych wyników wyszukiwania — skorzystajmy z polecenia `head -10`:

```
$ sudo docker search mysql | head -10
NAME                      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
mysql                     MySQL is a widely used, open-source relati...   2759      [OK]
mysql/mysql-server        Optimized MySQL Server Docker images. Crea...   178                  [OK]
centurylink/mysql         Image containing mysql. Optimized to be li...   46                   [OK]
sameersbn/mysql                                                           36                   [OK]
appcontainers/mysql       Centos/Debian Based Customizable MySQL Con...   8                    [OK]
marvambass/mysql          MySQL Server based on Ubuntu 14.04              6                    [OK]
alterway/mysql            Docker Mysql                                    2                    [OK]
drupaldocker/mysql        MySQL for Drupal                                2                    [OK]
azukiapp/mysql            Docker image to run MySQL by Azuki - http:...   2                    [OK]
$
```

Jak widzisz, wyniki wyszukiwania są sortowane według ich oceny (*STARS*). W wynikach wyszukiwania podawana jest informacja o tym, czy obraz jest oficjalny (utworzony i udostępniany przez oficjalne repozytorium Dockera). Ponadto podawana jest informacja dotycząca tego, czy obraz został zbudowany za pomocą struktury szkieletowej automatyzacji stworzonej przez twórców platformy Docker. Obraz *mysql* utworzony i udostępniony przez Docker Inc. uzyskał ocenę *2759* gwiazdek. Świadczy to o tym, że jest to najpopularniejszy obraz *mysql*. Polecamy korzystanie tylko z obrazów udostępnionych oficjalnie przez Docker Inc. z powodów bezpieczeństwa. Kolejnym obrazem na liście jest *mysql-server* udostępniony przez niezależnego użytkownika. Obraz ten uzyskał ocenę *178* gwiazdek. Kontenery Dockera są standardowymi blokami konstrukcyjnymi aplikacji rozproszonych.

Dynamiczne repozytorium obrazów Dockera utrzymuje się dzięki wsparciu entuzjastów z całego świata. Programiści mogą pobierać obrazy z serwisu Docker Hub, a także udostępniać w nim różne obrazy i kontenery zoptymalizowane pod kątem konkretnych potrzeb biznesowych. Dzięki temu można w dużym stopniu zautomatyzować i przyśpieszyć proces tworzenia, wdrażania i obsługi aplikacji. Docker Hub ma na celu zapewnienie doskonałej bazy obrazów dla programistów i administratorów systemu, dzięki której mogą oni skupić się na tworzeniu nowych funkcji poprzez zminimalizowanie powtarzalnej pracy nad podstawowymi elementami konstrukcji aplikacji.

Na podstawie zapytań wyszukiwarki Docker Hub Registry i rozmów z wieloma różnymi programistami twórcy Dockera doszli do wniosku, że programiści potrzebują gotowych stosów modułów utworzonych w preferowanych przez nich językach. Programistom zależy na maksymalnym skróceniu czasu pracy — pisaniu kodu bez tracenia czasu na walkę ze środowiskiem programistycznym, z elementami konstrukcyjnymi aplikacji i bibliotekami.

# Praca z interaktywnym kontenerem

W rozdziale 1. uruchomiłeś kontener *Hello World* i zobaczyłeś na własne oczy, jak działa technologia konteneryzacji. W tej sekcji uruchomisz kontener w trybie interaktywnym. Jako parametr polecenia `docker run` podaje się nazwę obrazu wejściowego, który jest następnie uruchamiany jako kontener. W celu uruchomienia kontenera w trybie interaktywnym należy do polecenia `docker run` dodać flagi `-t` i `-i`. Flaga `-i` sprawia, że kontener zostanie uruchomiony w trybie interaktywnym (przechwytuje ona standardowe wejście kontenera *STDIN*). Flaga `-t` alokuje emulator terminala, zwany również pseudoterminalem, a następnie przypisuje go do kontenera.

W poniższym przykładzie uruchomimy interaktywny kontener na podstawie obrazu `ubuntu:16.04` przy użyciu parametru `/bin/bash`:

```
$ sudo docker run -i -t ubuntu:16.04 /bin/bash
```

Nie pobrałeś jeszcze obrazu *ubuntu*, więc w terminalu zostanie wyświetlony komunikat, że ten obraz nie jest dostępny w lokalnym zbiorze obrazów, a następnie automatycznie uruchomi się proces pobierania tego obrazu:

```
Unable to find image 'ubuntu:16.04' locally
16.04: Pulling from library/ubuntu
```

Od razu po pobraniu obrazu zostanie on uruchomiony jako kontener. Ponadto wewnątrz kontenera uruchomiona zostanie powłoka Bash (dzieje się tak, ponieważ podczas uruchamiania polecenia podaliśmy argument `/bin/bash`). Bash wyświetli komunikat podobny do poniższego:

```
root@742718c21816:/#
```

Komunikat ten świadczy o poprawnym działaniu kontenera i gotowości do przyjęcia poleceń. Liczba szesnastkowa *742718c21816* to nazwa hosta kontenera. W Dockerze nazwa hosta jest taka sama jak identyfikator kontenera.

Spróbuj uruchomić kilka poleceń — sprawdź możliwość interaktywnej pracy z kontenerem:

```
root@742718c21816:/# hostname
742718c21816
root@742718c21816:/# id
uid=0(root) gid=0(root) groups=0(root)
root@742718c21816:/# echo $PS1
[e]0;u@h: wa]${debian_chroot:+($debian_chroot)}u@h:w$
root@742718c21816:/#
```

Na podstawie tych trzech poleceń widać, że znak gotowości do działania składa się z identyfikatora użytkownika, nazwy hosta i bieżącego katalogu.

Teraz skorzystajmy z jednej z niszowych funkcji Dockera — możliwości odczepienia Dockera od interaktywnego kontenera. Przyjrzyjmy się również szczegółom zarządzania tym kontenerem przez Dockera. Możemy odłączyć Dockera od kontenera za pomocą sekwencji kombinacji klawiszy *Ctrl+P* i *Ctrl+Q*. Sekwencja ta odłącza pseudoterminal od kontenera i przenosi użytkownika do znaku gotowości hosta Dockera ($), ale nie spowoduje to zakończenia pracy kontenera. W celu wyświetlenia listy wszystkich działających kontenerów i ich najważniejszych parametrów skorzystaj z polecenia `docker ps`:

```
$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED              STATUS
PORTS   NAMES
742718c21816   ubuntu:16.04   "/bin/bash"   About a minute ago   Up About a minute
        jolly_lovelace
```

Polecenie `docker ps` wyświetla tabelę z następującymi kolumnami:

- `CONTAINER ID` (identyfikator kontenera) — identyfikator przypisany kontenerowi. Do każdego kontenera przypisany jest unikalny identyfikator. Jest to losowa liczba szesnastkowa mająca 64 cyfry. Domyślnie polecenie `docker ps` wyświetla tylko 12 cyfr. W celu wyświetlenia wszystkich 64 cyfr skorzystaj z flagi `--no-trunc` (np. `sudo docker ps --no-trunc`).
- `IMAGE` (obraz) — obraz, na podstawie którego uruchomiono kontener.
- `COMMAND` (polecenie) — polecenie wywołane w momencie uruchomienia kontenera.
- `CREATED` (utworzony) — czas, który upłynął od utworzenia kontenera.
- `STATUS` (status) — bieżący status kontenera.
- `PORTS` (porty) — kolumna informująca o tym, czy do kontenera przypisano porty.
- `NAMES` (nazwy) — silnik Dockera automatycznie generuje losową nazwę kontenera składającą się z angielskiego przymiotnika i rzeczownika. W celu odwołania się do kontenera można używać jego identyfikatora lub nazwy. Nazwa kontenera może zostać nadana ręcznie za pomocą parametru `--name` dodanego do polecenia `docker run`.

Po przyjrzeniu się statusowi kontenera dołączmy go ponownie do Dockera. Skorzystaj z zaprezentowanego poniżej polecenia `docker attach`. Możesz posłużyć się nazwą kontenera lub jego identyfikatorem. W zaprezentowanym przykładzie wykorzystaliśmy nazwę kontenera. Jeżeli w oknie terminala nie widzisz znaku gotowości, ponownie wciśnij klawisz *Enter*:

```
$ sudo docker attach jolly_lovelace
root@742718c21816:/#
```

> Docker umożliwia wielokrotne podłączanie się do kontenera, co jest bardzo przydatne przy współdzieleniu ekranu.

Polecenie `docker attach` ponownie spowoduje wyświetlenie znaku gotowości kontenera. Poeksperymentuj z interaktywnym kontenerem — uruchom następujące polecenia:

```
root@742718c21816:/# pwd
/
root@742718c21816:/# ls
bin dev home lib64 mnt proc run srv tmp var
boot etc lib media opt root sbin sys usr
root@742718c21816:/# cd usr
root@742718c21816:/usr# ls
bin games include lib local sbin share src
root@742718c21816:/usr# exit
exit
$
```

Interaktywny kontener zostanie zatrzymany od razu po zakończeniu procesu Bash za pomocą polecenia `exit`. W związku z tym w oknie terminala ponownie zostanie wyświetlony znak gotowości Dockera (`$`).

## Śledzenie zmian wewnątrz kontenera

Podczas lektury poprzedniej sekcji dowiedziałeś się, jak utworzyć kontener oparty na bazowym obrazie *ubuntu*, i sprawdziłeś działanie podstawowych poleceń służących do podłączania i odłączania kontenerów. Poznałeś również polecenie `docker ps`, które wyświetla podstawowe dane niezbędne do zarządzania kontenerami. W tej sekcji pokażemy Ci, jak skutecznie śledzić zmiany wprowadzone w kontenerze i porównywać je z obrazem użytym do uruchomienia kontenera. Uruchom kontener w trybie interaktywnym, tak jak robiłeś to w poprzedniej sekcji:

```
$ sudo docker run -i -t ubuntu:16.04 /bin/bash
```

Przejdź do katalogu */home*:

```
root@d5ad60f174d3:/# cd /home
```

Teraz skorzystaj z polecenia `touch` w celu utworzenia trzech pustych plików. Za pierwszym razem, gdy użyjesz polecenia `ls -l`, zobaczysz, że katalog jest pusty. Drugie użycie tego polecenia pokaże trzy puste pliki utworzone za pomocą polecenia `touch`:

```
root@d5ad60f174d3:/home# ls -l
total 0
root@d5ad60f174d3:/home# touch {abc,cde,fgh}
root@d5ad60f174d3:/home# ls -l
total 0
-rw-r--r-- 1 root root 0 Sep 29 10:54 abc
-rw-r--r-- 1 root root 0 Sep 29 10:54 cde
-rw-r--r-- 1 root root 0 Sep 29 10:54 fgh
root@d5ad60f174d3:/home#
```

Docker Engine elegancko zarządza swoim systemem plików i pozwala na przeprowadzenie inspekcji systemu plików kontenera za pomocą polecenia `docker diff`. W celu przeprowadzenia inspekcji systemu plików kontenera należy go odczepić lub otworzyć inne okno terminala z sesją hosta Dockera, a następnie uruchomić polecenie `docker diff`. Wiemy, że dowolny kontener *ubuntu* ma nazwę hosta, która jest częścią znaku gotowości. Nazwa ta jest również identyfikatorem kontenera. Możemy z niej skorzystać, uruchamiając polecenie `docker diff`:

```
$ sudo docker diff d5ad60f174d3
```

W zaprezentowanym przykładzie uruchomienie polecenia `docker diff` wygeneruje cztery linie odpowiedzi:

```
C /home
A /home/abc
A /home/cde
A /home/fgh
```

Odpowiedź ta oznacza, że katalog */home* został zmodyfikowany (świadczy o tym litera *C*) i utworzono pliki */home/abc*, */home/cde* i */home/fgh*, o czym świadczy litera *A*. Operacje usuwania oznaczane są literą *D*. Nie usuwaliśmy żadnych plików, a więc oznaczenie to nie pojawia się w wygenerowanych danych wyjściowych.

> Jeżeli podczas pracy z obrazem nie określimy konkretnego egzemplarza obrazu np. za pomocą nazwy, to Docker Engine będzie się odwoływać do ostatnio wygenerowanego egzemplarza (zostanie użyty parametr `latest`).

## Zarządzanie kontenerami

Dotychczas zaprezentowaliśmy kilka praktycznych przykładów ilustrujących sedno pracy z kontenerami. W tej sekcji przedstawimy kilka podstawowych, a także kilka bardziej zaawansowanych poleceń umożliwiających zarządzanie kontenerami.

Docker Engine pozwala na uruchomienie (`docker start`), zatrzymanie (`docker stop`) i ponowne uruchomienie (`docker restart`) kontenera. Zacznijmy od polecenia `docker stop` zatrzymującego aktywny kontener. Uruchomienie tego polecenia przez użytkownika sprawia, że Docker Engine wysyła sygnał `SIGTERM (-15)` do głównego procesu uruchomionego wewnątrz kontenera. Sygnał ten wywołuje jego poprawne zakończenie. Większość procesów obsługuje ten sygnał i zamyka się w sposób poprawny. Ale jeśli do tego nie dojdzie, Docker Engine odczekuje pewien czas i jeżeli w ciągu tego czasu proces nie zostanie zakończony, to Docker Engine zakończy ten proces metodą siłową. Siłowe zakończenie jest przeprowadzane poprzez wysłanie sygnału `SIGKILL(-9)`, który nie może być zignorowany. Powoduje on natychmiastowe zakończenie procesu bez przeprowadzenia sprzątania.

Czas uruchomić kontener i poeksperymentować z poleceniem `docker stop`:

```
$ sudo docker run -i -t ubuntu:16.04 /bin/bash
root@da1c0f7daa2a:/#
```

Uruchomiliśmy kontener. Teraz czas skorzystać z polecenia `docker stop`. Do polecenia musisz dodać argument w postaci identyfikatora kontenera (skopiuj go ze znaku gotowości). Oczywiście polecenie to należy wprowadzić w drugim oknie lub instancji terminala. Polecenie to zawsze wyświetla identyfikator kontenera:

```
$ sudo docker stop da1c0f7daa2a
da1c0f7daa2a
```

Jeżeli przejdziesz do okna lub instancji terminala, w której uruchomiłeś wcześniej kontener, to zauważysz, że działanie kontenera zostało zakończone. Jeżeli przyjrzysz się uważniej zawartości tego okna, zauważysz napis *exit* znajdujący się obok znaku gotowości kontenera. Wynika to z obsługi sygnału `SIGTERM` przez powłokę Bash:

```
root@da1c0f7daa2a:/# exit
$
```

Gdybyśmy poszli krok dalej i uruchomili polecenie `docker ps`, kontener nie znalazłby się na wyświetlonej liście. Wynika to z tego, że polecenie `docker ps` wyświetla listę zawierającą tylko kontenery znajdujące się w stanie aktywnym. Nasz kontener znalazł się w stanie zatrzymania, a więc nie został ujęty na tej liście. Jak w związku z tym zobaczyć kontenery, które znajdują się w stanie zatrzymania? Polecenie `docker ps` może przyjąć dodatkowy argument `-a`, który spowoduje wyświetlenie wszystkich kontenerów hosta Dockera niezależnie od ich statusu:

```
$ sudo docker ps -a
CONTAINER ID   IMAGE   COMMAND
CREATED        STATUS  PORTS
NAMES
da1c0f7daa2a           ubuntu:16.04               "/bin/bash"
20 minutes ago         Exited (0) 10 minutes ago
desperate_engelbart
$
```

Przyjrzyjmy się poleceniu `docker start`. Służy ono do uruchamiania jednego wstrzymanego kontenera lub wielu takich kontenerów. Kontener może być wprowadzony w stan zatrzymania za pomocą polecenia `docker stop` lub w wyniku zakończenia głównego procesu kontenera (może do tego dojść w sposób naturalny lub w wyniku błędu). Polecenie to nie wpływa na pracę aktywnych kontenerów.

Uruchom zatrzymany wcześniej kontener za pomocą polecenia `docker start` (jako argument tego polecenia musisz podać identyfikator kontenera):

```
$ sudo docker start da1c0f7daa2a
da1c0f7daa2a
$
```

Domyślnie polecenie `docker start` nie wywoła podłączenia kontenera. W celu podłączenia kontenera należy uruchomić polecenie `docker start` z parametrem `-a` lub polecenie `docker attach` w sposób jawny:

```
$ sudo docker attach da1c0f7daa2a
root@da1c0f7daa2a:/#
```

Teraz możesz uruchomić polecenie `docker ps` i zweryfikować status pracy kontenera:

```
$ sudo docker ps
CONTAINER ID          IMAGE           COMMAND
CREATED               STATUS          PORTS
NAMES
da1c0f7daa2a          ubuntu:16.04    "/bin/bash"
25 minutes ago        Up 3 minutes
desperate_engelbart
$
```

Polecenie `restart` łączy funkcje poleceń `stop` i `start`. Innymi słowy: polecenie `restart` zatrzyma działanie kontenera — wykona dokładnie te same operacje co polecenie `docker stop`, a następnie zainicjuje proces uruchamiania (`start`). Operacje te są wykonywane domyślnie po uruchomieniu polecenia `docker restart`.

Kolejną przydatną parą poleceń jest `docker pause` i `docker unpause`. Polecenie `docker pause` wstrzymuje wykonywanie wszystkich procesów w kontenerze. Polecenie `docker unpause` spowoduje ponowienie wykonywania wszystkich procesów kontenera od punktu, w którym zostały one wstrzymane.

Czas zobaczyć, jak w praktyce działają polecenia `docker pause` i `docker unpause`. Stworzyliśmy przykład korzystający z dwóch okien terminala. W jednym z nich uruchomiliśmy zaprezentowany wcześniej kontener *ubuntu*, a następnie wywołaliśmy nieskończoną pętlę wyświetlającą datę i czas w odstępach co 5 s. Aby samodzielnie wykonać ten eksperyment, uruchom następujące polecenia:

```
$ sudo docker run -i -t ubuntu:16.04 /bin/bash
root@c439077aa80a:/# while true; do date; sleep 5; done
Thu Oct 2 03:11:19 UTC 2016
```

```
Thu Oct 2 03:11:24 UTC 2016
Thu Oct 2 03:11:29 UTC 2016
Thu Oct 2 03:11:34 UTC 2016
Thu Oct 2 03:11:59 UTC 2016
Thu Oct 2 03:12:04 UTC 2016
Thu Oct 2 03:12:09 UTC 2016
Thu Oct 2 03:12:14 UTC 2016
Thu Oct 2 03:12:19 UTC 2016
Thu Oct 2 03:12:24 UTC 2016
Thu Oct 2 03:12:29 UTC 2016
Thu Oct 2 03:12:34 UTC 2016
```

Nasz skrypt wyświetlał datę i bieżący czas dokładnie co 5 s z jednym wyjątkiem:

```
Thu Oct 2 03:11:34 UTC 2016
Thu Oct 2 03:11:59 UTC 2016
```

Tutaj widoczne jest upłynięcie 25 s. Wynika ono z uruchomienia polecenia `docker pause` odwołującego się do tego kontenera w drugim oknie terminala:

```
$ sudo docker pause c439077aa80a
c439077aa80a
```

Po wstrzymaniu kontenera sprawdziliśmy status jego procesu za pomocą polecenia `docker ps` (polecenie to uruchomiliśmy również w drugim oknie terminala). Oczywiście zwrócone dane poinformowały o wstrzymaniu pracy kontenera (status *Paused*):

```
$ sudo docker ps
CONTAINER ID     IMAGE                  COMMAND
CREATED          STATUS                 PORTS          NAMES
c439077aa80a     ubuntu:16.04           "/bin/bash"
47 seconds ago   Up 46 seconds (Paused)                ecstatic_torvalds
```

Następnie uruchomiliśmy polecenie `docker unpause`, które ponowiło działanie wstrzymanego wcześniej kontenera. Kontener zaczął ponownie wyświetlać datę i czas:

```
$ sudo docker unpause c439077aa80a
c439077aa80a
```

Podczas lektury tej sekcji poznałeś polecenia `docker pause` i `docker unpause`. Teraz czas zatrzymać kontener za pomocą polecenia `docker stop`:

```
$ sudo docker stop c439077aa80a
c439077aa80a
```

## Administrowanie kontenerami

W poprzednich przykładach po uruchomieniu polecenia `docker ps -a` widać było wiele zatrzymanych kontenerów. Kontenery te mogłyby wisieć w stanie zatrzymania przez wieki. Może się to wydawać błędem, ale w praktyce pozwala na utworzenie obrazu na podstawie takiego kontenera,

a także na ponowne uruchomienie wybranych zatrzymanych kontenerów. Jednakże nie wszystkie zatrzymane kontenery będą używane ponownie, a każdy z takich kontenerów zajmuje przestrzeń dyskową systemu plików hosta Dockera. Docker Engine oferuje kilka sposobów rozwiązania tego problemu. Przyjrzyjmy się im.

Podczas uruchamiania kontenera możemy poprosić Docker Engine, aby sprzątnął kontener po osiągnięciu stanu zatrzymania. W tym celu należy zastosować dodany do polecenia `docker run` parametr `--rm` (`sudo docker run -i -t --rm ubuntu:16.04 /bin/bash`).

Innym rozwiązaniem tego problemu jest wyświetlenie listy wszystkich kontenerów (polecenie `docker ps -a`) i ręczne usunięcie zbędnych zatrzymanych kontenerów za pomocą polecenia `docker rm`:

```
$ sudo docker ps -a
CONTAINER ID     IMAGE                     COMMAND
CREATED          STATUS                    PORTS          NAMES
7473f2568add     ubuntu:16.04              "/bin/bash"
5 seconds ago    Exited (0) 3 seconds ago                 jolly_wilson

$ sudo docker rm 7473f2568add
7473f2568add
$
```

Polecenia `docker rm` i `docker ps` można połączyć w celu automatycznego usunięcia wszystkich zatrzymanych kontenerów:

```
$ sudo docker rm $(sudo docker ps -aq)
```

Polecenie znajdujące się w nawiasie `$()` wygeneruje listę identyfikatorów wszystkich kontenerów niezależnie od ich statusu. Lista ta stanie się argumentem polecenia `docker rm`, które jeżeli nie zostanie uruchomione z parametrem `-f`, usunie tylko te kontenery, które nie są aktywne. Polecenie to po trafieniu na uruchomiony kontener zwróci poniższy komunikat informujący, że nie można usunąć aktywnego kontenera, a następnie przejdzie do kolejnego kontenera z listy:

```
Error response from daemon: You cannot remove a running container.
Stop the container before attempting removal or use -f
```

Możesz uniknąć takich sytuacji poprzez przefiltrowanie listy i wybranie tylko kontenerów w stanie `exited` za pomocą dodanego do polecenia `docker ps` parametru `-f`:

```
$ sudo docker rm $(sudo docker ps -aq -f state=exited)
```

Irytuje Cię wpisywanie tak długich i skomplikowanych poleceń? Mamy dla Ciebie dobre wieści. Wszystkie zatrzymane kontenery można usunąć za pomocą polecenia `docker container prune`. Wprowadzono je do Dockera w wersji 1.13. Oto przykład uruchomienia polecenia `docker container prune`:

```
$ sudo docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
9b1aaaf108d3922d1a503fe01e9024302f0434a3b387c450d3b302020966a13e
d43c75065c6147501a7bc62f418fe501eeabaddd8617d77a4b28b5807dfeaa89
1614c44092f1c358cbb248a49430e70b674b52b32b8a193da9bba9b7136d1640

Total reclaimed space: 0 B
```

## Budowanie obrazów na podstawie kontenerów

Dotychczas pracowaliśmy na kontenerach powstałych na bazie standardowych obrazów *busybox* i *ubuntu*. W tej sekcji dowiesz się, jak dołączyć dodatkowe oprogramowanie do obrazu bazowego poprzez konwersję aktywnego kontenera na obraz przeznaczony do dalszego użytku.

W roli bazowego obrazu zastosujmy *ubuntu:16.04*. Dodajmy do niego aplikację *wget*, a następnie przekształćmy działający kontener na obraz. W tym celu należy wykonać następujące czynności:

1. Uruchom kontener *ubuntu:16.04* za pomocą polecenia `docker run`:

   ```
   $ sudo docker run -i -t ubuntu:16.04 /bin/bash
   ```

2. Po uruchomieniu kontenera zweryfikuj obecność w wybranym obrazie aplikacji *wget*. Skorzystaj z polecenia `which` z argumentem `wget`. Polecenie to zwróci pusty wiersz, co oznacza, że aplikacja *wget* nie została zainstalowana w tym kontenerze:

   ```
   root@472c96295678:/# which wget
   root@472c96295678:/#
   ```

3. Czas zainstalować aplikację *wget*. Pracujesz z nowym kontenerem *ubuntu*, więc przed rozpoczęciem instalacji pakietu *wget* musisz dokonać synchronizacji z repozytorium pakietów *ubuntu*:

   ```
   root@472c96295678:/# apt-get update
   ```

4. Po zakończeniu synchronizacji z repozytorium pakietów możesz rozpocząć instalację aplikacji *wget*:

   ```
   root@472c96295678:/# apt-get install -y wget
   ```

5. Po zakończeniu instalacji aplikacji *wget* upewnij się, że została ona zainstalowana poprawnie — uruchom polecenie `which` z argumentem `wget`:

   ```
   root@472c96295678:/# which wget
   /usr/bin/wget
   root@472c96295678:/#
   ```

6. Instalacja dowolnego oprogramowania spowoduje zmianę kontenera w stosunku do jego obrazu bazowego. Możemy to sprawdzić za pomocą polecenia `docker diff` opisanego w sekcji „Śledzenie zmian wewnątrz kontenera”. Uruchom je w drugim terminalu:

   ```
   $ sudo docker diff 472c96295678
   ```

W wyniku uruchomienia tego polecenia na ekranie zostanie wyświetlone kilkaset linii informujących o modyfikacji obrazu *ubuntu*. Modyfikacje te powstały w wyniku aktualizacji repozytorium pakietów, pobrania danych binarnych aplikacji *wget*, a także plików niezbędnych do jej obsługi.

7. Teraz czas na wykonanie najważniejszej czynności — zatwierdzenia obrazu. Polecenie `docker commit` może być uruchomione na aktywnym lub zatrzymanym kontenerze. W przypadku uruchomienia go na działającym kontenerze Docker Engine wstrzyma działanie kontenera podczas operacji `commit` w celu zapewnienia spójności danych. Mimo to zalecamy wykonywanie operacji `commit` tylko na zatrzymanym kontenerze. Oto przykład działania polecenia `docker commit`:

```
$ sudo docker commit 472c96295678 \
   learningdocker/ubuntu_wget
sha256:a530f0a0238654fa741813fac39bba2cc14457aee079a7ae1fe1c64dc7e1ac25
```

Zatwierdziliśmy obraz pod nazwą *learningdocker/ubuntu_wget*.

Teraz już wiesz, jak utworzyć obraz na podstawie zawartości kontenera. Wyświetl listę obrazów dostępnych w hoście Dockera i zobaczmy, czy znajdzie się na niej nowy obraz:

```
$ sudo docker images
REPOSITORY                    TAG             IMAGE ID
CREATED                       VIRTUAL SIZE
learningdocker/ubuntu_wget    latest          a530f0a02386
48 seconds ago                221.3 MB
busybox                       latest          e72ac664f4f0
2 days ago                    2.433 MB
ubuntu                        16.04           6b4e8a7373fe
2 days ago                    194.8 MB
```

Z danych zwróconych przez polecenie `docker images` wynika, że operacja tworzenia obrazu na podstawie kontenera przebiegła pomyślnie.

Umiesz już tworzyć obrazy na podstawie kontenerów. Chcielibyśmy zachęcić Cię, abyś korzystał z tej metody głównie podczas testowania. Najbardziej elegancką i polecaną metodą tworzenia obrazów jest metoda `Dockerfile`, która zostanie opisana w kolejnym rozdziale.

## Uruchamianie kontenera jako demona

Eksperymentowałeś już z interaktywnym kontenerem, śledziłeś zmiany dokonane w kontenerach, tworzyłeś obrazy na podstawie kontenerów, a więc zgłębiłeś pewne zagadnienia związane z konteneryzacją. Teraz czas, abyś zapoznał się z prawdziwym konikiem technologii Dockera. Tak jest! W tej sekcji dowiesz się, jak uruchomić kontener w trybie odłączonym — nauczysz się uruchamiać kontener jako demona. Ponadto zobaczysz, jak podglądać tekst wygenerowany w kontenerze.

Polecenie `docker run` obsługuje parametr `-d`, który służy do uruchamiania kontenera w trybie odłączonym, tj. uruchamiania go jako demona. W celu zilustrowania tej koncepcji wrócimy do przykładu zaprezentowanego wcześniej podczas wyjaśniania działania operacji `pause` i `unpause`:

```
$ sudo docker run -d ubuntu \
   /bin/bash -c "while true; do date; sleep 5; done"
0137d98ee363b44f22a48246ac5d460c65b67e4d7955aab6cbb0379ac421269b
```

Polecenie `docker logs` otwiera podgląd danych generowanych przez kontener demona:

```
$ sudo docker logs \
   0137d98ee363b44f22a48246ac5d460c65b67e4d7955aab6cbb0379ac421269b
Sat Oct 4 17:41:04 UTC 2016
Sat Oct 4 17:41:09 UTC 2016
Sat Oct 4 17:41:14 UTC 2016
Sat Oct 4 17:41:19 UTC 2016
```

# Podsumowanie

W tym rozdziale poznałeś różne zagadnienia związane z działaniem kontenerów Dockera. Na początku rozdziału wyjaśniliśmy ważne terminy, takie jak: obraz, kontener, rejestr i repozytorium, co ułatwiło zrozumienie koncepcji przedstawionych w kolejnych podrozdziałach. Dowiedziałeś się, jak szukać obrazów w repozytorium Dockera. Ponadto poznałeś różne zagadnienia związane z działaniem i obsługą kontenerów Dockera. Potrafisz śledzić zmiany dokonywane wewnątrz kontenera i nimi administrować.

Podczas lektury kolejnego rozdziału poznasz techniki i narzędzia przydatne w procesie budowania obrazów Dockera.

# Skorowidz

## E



## F



## G

## H



## I



## J



## K

## L



## M



## N



## O



## P

## Q



## R



## S, Ś

## T



## U



## V



## W



## Z

## Docker — nowy wymiar rozwoju aplikacji!

Docker to platforma oparta na otwartym kodzie źródłowym. Jest dziś uważany za najważniejsze narzędzie do rozwijania aplikacji rozproszonych. Umożliwia przyspieszenie prac nad oprogramowaniem, ale również jego konfiguracją, wdrożeniem i dostarczeniem do klienta. Docker pozwala na skalowanie infrastruktury aplikacji w czasie rzeczywistym i poprawę wykorzystania zasobów. Technologia Dockera ułatwia budowę aplikacji dobrze przemyślanych, przetwarzających dane w sposób kontekstowy, rozproszonych i skupionych na konkretnym celu biznesowym. Co więcej, pozwala na uruchamianie tworzonej aplikacji na dowolnej platformie!

Niniejsza książka jest przeznaczona dla programistów i osób zarządzających procesem tworzenia aplikacji. Przedstawiono tu informacje potrzebne do rozpoczęcia pracy z Dockerem, wyjaśniono też terminologię i polecenia potrzebne do zarządzania kontenerami. Opisano proces budowy i publikacji obrazów Dockera, nie zabrakło również informacji o sposobach tworzenia prywatnych repozytoriów. W książce poruszono także zagadnienia związane z orkiestracją kontenerów za pomocą narzędzia Docker Compose, debugowaniem kontenerów i zabezpieczaniem ich przy użyciu modułów AppArmor i SELinux. Ciekawym elementem jest opis kilku praktycznych zastosowań platformy, dzięki czemu łatwiej uświadomić sobie przyczyny rosnącej popularności tego rozwiązania.

**W tej książce między innymi:**

- rozpoczęcie pracy z Dockerem
- automatyczne budowanie obrazów Dockera
- udostępnianie danych i związane z tym problemy
- testowanie i debugowanie aplikacji
- zagadnienia bezpieczeństwa i zarządzania konfiguracją Dockera

**Jeeva S. Chelladhurai** — działa w branży IT od 20 lat. Specjalizuje się w DevOps i dostarczaniu rozwiązań mających postać chmury. Interesuje się centrami optymalizacji danych i rozwijaniem aplikacji za pomocą Dockera.
**Vinod Singh** — od wielu lat tworzy oprogramowanie, projektuje architekturę systemów i testuje różne rozwiązania. Jego pasją jest tworzenie oprogramowania w chmurze, rozwój sztucznej inteligencji i system Linux.
**Dr Pethuru Raj** — jest głównym architektem w Reliance Jio Cloud, wcześniej był architektem infrastruktury chmury w indyjskim IBM Global Cloud Center of Excellence. W IT pracuje od ponad 17 lat, a od 8 lat zajmuje się również badaniami naukowymi.


ISBN 978-83-283-3972-9
9 788328 339729
Cena: 49,00 zł
Packt