Russ McKendrick, Scott Gallagher

Docker

Programowanie aplikacji dla zaawansowanych

Wydanie II

Helion

Packt›

Tytuł oryginału: Mastering Docker - Second Edition

Tłumaczenie: Konrad Matuk

ISBN: 978-83-283-4308-5

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/dockaz
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/dockaz.zip

Printed in Poland.

Spis treści

1

ROZDZIAŁ

Docker — wprowadzenie

Witaj w drugim wydaniu książki *Docker. Programowanie aplikacji dla zaawansowanych*. W rozdziale 1. znajdziesz podstawowe wiadomości dotyczące Dockera, które powinieneś opanować przed przystąpieniem do lektury kolejnych rozdziałów. Jeżeli nie masz zbyt dużego doświadczenia w pracy z Dockerem, to nie przejmuj się, po przeczytaniu tego rozdziału dowiesz się wszystkiego, co niezbędne, aby pójść dalej. Po przeczytaniu całej książki powinieneś zostać mistrzem Dockera potrafiącym implementować Dockera we własnych środowiskach, będącym w stanie zbudować i uruchomić aplikacje w kontenerach.

W tym rozdziale opiszemy następujące wysokopoziomowe zagadnienia:

- Czym jest Docker?
- Czym się różni Docker od standardowych maszyn wirtualnych?
- Jak wygląda proces instalacji Dockera?
- Jakie polecenia obsługuje Docker?
- Jak wygląda ekosystem Dockera?

Czym jest Docker?

Definiowanie, czym jest Docker, zacznijmy od cytatu ze strony internetowej tej platformy:

> *Docker to obecnie najpopularniejsza platforma przeznaczona do konteneryzacji aplikacji. Umożliwia ona rozwiązanie problemów typu „a u mnie działa" powstających podczas tworzenia aplikacji przez różnych programistów. Administratorzy korzystają z Dockera w celu uruchamiania na jednym serwerze wielu aplikacji w oddzielonych od siebie*

kontenerach i zarządzania nimi. Docker jest używany w wielu firmach podczas pracy zgodnie z metodologią programowania zwinnego, umożliwiającej szybsze tworzenie nowych funkcji, a także zwiększenie bezpieczeństwa w aplikacjach serwerowych systemów Linux i Windows.

Te słowa nie oddają wszystkiego. Oto liczby opisujące popularność Dockera przedstawione przez prezesa firmy będącej twórcą Dockera, Bena Goluba, podczas otwarcia konferencji DockerCon w 2017 r.:

- 14 milionów hostów Dockera;
- 900 tysięcy aplikacji korzystających z Dockera;
- wzrost o 77 000% liczby miejsc pracy związanych z Dockerem;
- 12 miliardów pobrań obrazów;
- 3300 osób zaangażowanych w rozwój platformy.

Te liczby robią wrażenie, jeżeli weźmiemy pod uwagę, że Docker to technologia obecna na rynku dopiero od trzech lat. Materiał wideo z tym wystąpieniem możesz obejrzeć na stronie *https://www.slideshare.net/Docker/dockercon-2017-general-session-day-1-ben-golub*.

Spróbujmy zrozumieć problemy, które rozwiązuje Docker. Zacznijmy od problemów programistów.

Programiści

Poniższy mem to chyba najlepsza ilustracja problemu „na moim komputerze działała bez problemów". Obraz ten zaczął kilka lat temu pojawiać się na prezentacjach, forach dyskusyjnych i w kanałach Slacka:

To dość śmieszna ilustracja częstego problemu, z którym spotykałem się osobiście.

Problem

Pomimo zachowania najlepszych praktyk metodologii DevOps wciąż możliwe jest istnienie różnic pomiędzy środowiskiem programistycznym i produkcyjnym.

Programista korzystający z wersji PHP przystosowanej do pracy w systemie macOS prawdopodobnie będzie pracował w innej wersji niż serwer produkcyjny pracujący pod kontrolą systemu Linux. Nawet jeżeli obie wersje PHP są identyczne, to środowiska te mogą charakteryzować się inną konfiguracją, innymi przywilejami dotyczącymi korzystania z plików itd. Oto jedno potencjalne źródło problemu.

Programista wdrażający kod w środowisku produkcyjnym musi sobie odpowiedzieć na następujące pytanie: Czy lepiej jest skonfigurować środowisko produkcyjne tak, aby jego konfiguracja odpowiadała środowisku deweloperskiemu, czy lepiej skonfigurować środowisko deweloperskie tak, aby było skonfigurowane tak samo jak środowisko produkcyjne?

Idealnie byłoby, aby wszystko było skonfigurowane w identyczny sposób. Dotyczy do laptopa programisty i produkcyjnego serwera, ale to dość utopijna wizja, którą trudno zrealizować. Każdy pracuje w nieco inny sposób. Zgodność platform trudno jest uzyskać nawet wtedy, gdy wszystko obsługuje ta sama osoba, a w przypadku zespołu składającego się z setki programistów jest to praktycznie niemożliwe.

Rozwiązanie problemu za pomocą Dockera

Programista korzystający z platformy Docker przeznaczonej dla systemu Windows lub Mac może obudować swój kod kontenerem, który może być zdefiniowany samodzielnie albo utworzony za pomocą pliku Dockerfile (zagadnienie to opiszemy w rozdziale 2. „Tworzenie obrazów kontenerów") lub pliku Docker Compose (zagadnienie to opiszemy w rozdziale 6. „Docker Compose").

Dzięki temu rozwiązaniu możliwe jest korzystanie z wybranego środowiska programistycznego i preferowanych sposobów pracy nad kodem. Podczas lektury kolejnych rozdziałów przekonasz się o tym, że instalacja i obsługa Dockera wcale nie są trudne. Korzystanie z niego jest o wiele łatwiejsze od dbania o zgodność środowisk nawet za pomocą mechanizmów automatyzacji.

Administratorzy

Pracę administratora wykonywałem dłużej, niż powinienem, i z mojego doświadczenia wynika, że opisany niżej problem pojawia się dość regularnie.

Problem

Załóżmy, że opiekujesz się pięcioma serwerami: trzy z nich to serwery sieciowe o dość zrównoważonym obciążaniu, a dwa pozostałe to serwery bazodanowe działające w konfiguracji serwer nadrzędny i serwer podrzędny, na których uruchomiona jest aplikacja 1. Stosem oprogramowania

zarządzasz automatycznie za pomocą narzędzia takiego jak Puppet lub Chef. Narzędzie to jest również używane do konfiguracji wszystkich pięciu serwerów.

Wszystko działało poprawnie, dopóki nie dostałeś zadania wdrożenia aplikacji 2 na tych samych serwerach, na których działa aplikacja 1. To na pozór nic problematycznego. Możesz zmodyfikować konfigurację narzędzia Puppet lub Chef w celu dodania nowych użytkowników i wirtualnych hostów oraz zaciągnięcia nowego kodu, ale po chwili orientujesz się, że aplikacja 2 wymaga wyższej wersji oprogramowania od aplikacji 1.

Okazuje się, że aplikacja 1 nie chce współpracować z nowym stosem oprogramowania, a aplikacja 2 nie jest wstecznie kompatybilna.

Istnieje kilka tradycyjnych rozwiązań tego problemu, ale żadne z nich nie jest idealne:

1. Możesz poprosić o dodatkowe serwery. Technicznie rzecz biorąc, jest to najbezpieczniejsze rozwiązanie, ale wiąże się to z poniesieniem wydatków na dodatkową infrastrukturę.
2. Możesz zmienić architekturę aplikacji. Wyjęcie jednego z serwerów sieciowych i bazodanowych będącego kopią lub poprawiającego obciążenia robocze i wdrożenie na nim stosu aplikacji 2 może wydawać się również dobrym rozwiązaniem. Jednakże sprawi ono, że aplikacja 2 będzie podatna na awarie, a stabilność pracy aplikacji 1 może ulec pogorszeniu (prawdopodobnie z jakiegoś powodu działała ona na trzech serwerach sieciowych i dwóch serwerach bazodanowych).
3. Możesz podjąć próbę równoległej instalacji nowego stosu oprogramowania. Oczywiście jest to możliwe i może wydawać się dobrym tymczasowym rozwiązaniem umożliwiającym uruchomienie nowego projektu, ale wdrożenie krytycznej poprawki zabezpieczeń któregoś ze stosów może być wtedy bardzo kłopotliwe.

Rozwiązanie problemu za pomocą Dockera

Zastosowanie Dockera to doskonałe rozwiązanie tego problemu. Gdyby aplikacja 1 była uruchomiona na trzech serwerach w kontenerach, to mógłbyś nawet zdublować te kontenery, co pozwoliłoby na gładkie wdrażanie nowych wersji tej aplikacji „w locie", bez ograniczania jej dostępności.

Wdrożenie aplikacji 2 w tym środowisku sprowadza się po prostu do uruchomienia większej liczby kontenerów w trzech hostach i skonfigurowania systemu równoważenia obciążenia. Wdrażając kontenery, nie musisz się przejmować problemami wynikającymi z wdrażania, konfigurowania i zarządzania dwiema wersjami tego samego stosu oprogramowania na tym samym serwerze.

Dokładnie taki scenariusz opiszemy w rozdziale 6. „Docker Compose".

Firmy

Firmy cierpią z powodu opisanych wcześniej problemów, ponieważ zatrudniają programistów i administratorów, ale problemy te w skali firmy wydają się znacznie większe i wiąże się z nimi większe ryzyko.

Problem

Przerwa w działaniu aplikacji wiąże się z poniesieniem kosztów finansowych, a także utratą reputacji. Firmy muszą więc testować każde wdrożenie przed jego udostępnieniem. W związku z tym nowe funkcje i poprawki są zawieszone z powodu:

- Tworzenia i konfiguracji środowisk testowych.
- Wdrażania aplikacji w nowych środowiskach.
- Wykonywania zaplanowanych testów i poprawiania aplikacji i konfiguracji aż do poprawnego przejścia testów.
- Zmian i omawiania żądań modyfikacji aplikacji podczas wdrażania jej kodu w środowisku produkcyjnym.

Proces ten może trwać od kilku dni do tygodni, a nawet miesięcy. Zależy to od złożoności aplikacji i ryzyka wprowadzanego przez zmiany. Proces ten jest niezbędny do zapewnienia ciągłości i dostępności na poziomie technologicznym, ale wiąże się z potencjalnymi zagrożeniami natury biznesowej. Co, jeżeli konkurencja udostępni podobne lub, o zgrozo, identyczne funkcje przed Tobą?

Takie zdarzenie może mieć tak samo negatywny wpływ na sprzedaż i reputację firmy jak udostępnienie nieprzetestowanego wdrożenia.

Rozwiązanie problemu za pomocą Dockera

Zaczniemy od stwierdzenia, że Docker nie zwalnia z wykonywania opisanego wcześniej procesu, ale jak już pisaliśmy, ułatwia wykonywanie pracy w sposób ciągły. Dzięki niemu programiści mogą pracować w tak samo skonfigurowanym kontenerze jak kontener wdrożony na produkcji, co z pewnością uprości metodologię jego testowania.

Gdy np. programista ładuje kod, który według niego działa w jego lokalnym środowisku (środowisku, w którym przeprowadzano prace programistyczne), to możliwe jest takie skonfigurowanie narzędzia testującego, aby uruchomiło te same kontenery w celu przeprowadzenia automatycznego testu. Po zakończeniu testowania kontenery mogą zostać usunięte w celu zwolnienia zasobów dla kolejnego zestawu testów. Technika ta sprawia, że proces testowania i związane z nim procedury są o wiele bardziej elastyczne i możliwe staje się wielokrotne korzystanie z tego samego środowiska, co zwalnia z konieczności ponownego wdrażania lub odtwarzania serwerów z obrazów przed przeprowadzeniem kolejnego zestawu testów.

Opisane rozwiązanie usprawnia proces wdrażania nowych aplikacji w środowisku produkcyjnym.

Im szybciej zostanie on przeprowadzony, tym wcześniej będziesz mógł pewnie wprowadzić nowe funkcje lub poprawki, nie dając się wyprzedzić konkurencji.

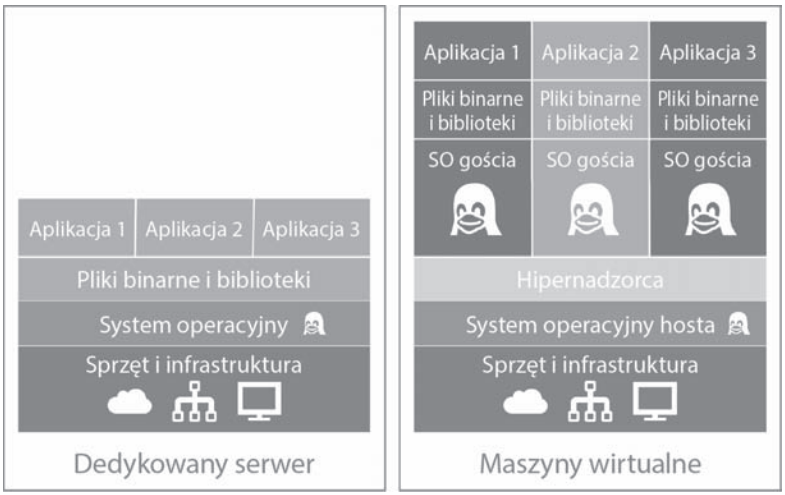
Różnice pomiędzy dedykowanymi hostami, maszynami wirtualnymi i Dockerem

Opisaliśmy problemy, które Docker miał rozwiązać. Teraz zajmiemy się opisem szczegółów mechanizmu działania tej platformy.

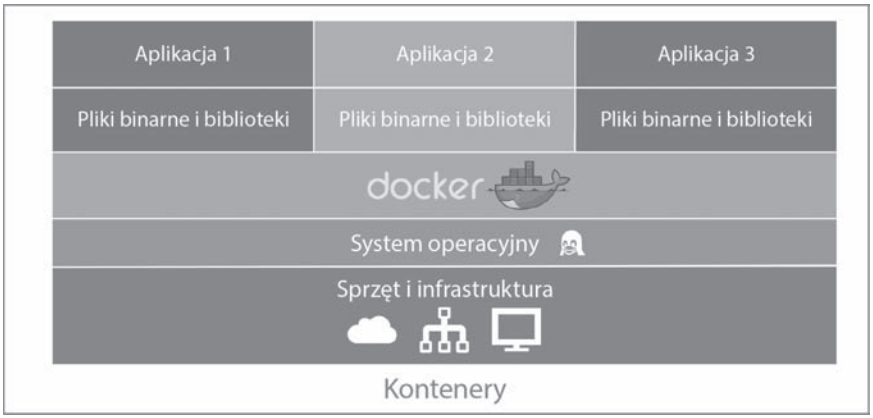
Docker to system zarządzania, który ułatwia obsługę kontenerów **LXC** (**Linux Containers**). Rozwiązanie to umożliwia tworzenie obrazów w środowiskach wirtualnych na laptopach programistów i uruchamianie na nich poleceń. Docker umożliwia wykonywanie na kontenerach uruchomionych lokalnie tych samych operacji, które będą przeprowadzane w środowisku produkcyjnym.

W związku z tym nie ma konieczności robienia czegoś w sposób inny w środowisku lokalnym i w środowisku serwerowym. Porównajmy różnice pomiędzy kontenerami Dockera i typowymi środowiskami maszyn wirtualnych.

Poniższy schemat przedstawia różnice pomiędzy dedykowanym, fizycznym serwerem i serwerem uruchomionym w maszynie wirtualnej:



Jak widzisz, w przypadku dedykowanego serwera wszystkie trzy aplikacje współdzielą ten sam stos oprogramowania wyróżniony kolorem pomarańczowym. Maszyny wirtualne umożliwiają uruchomienie trzech aplikacji na dwóch zupełnie innych stosach oprogramowania. Na poniższym schemacie pokazano te same aplikacje (wyróżnione kolorem pomarańczowym i zielonym) uruchomione w kontenerach Dockera:



Schemat ten przedstawia największą zaletę Dockera — brak potrzeby uruchamiania całego nowego systemu operacyjnego podczas dodawania nowych kontenerów, co pozwala na zredukowanie ich rozmiaru. Docker korzysta z jądra systemu Linux hosta. Większość wersji systemu Linux korzysta ze standardowych modeli jądra, a więc Docker może działać w dystrybucjach takich jak Red Hat, CentOS i Ubuntu. W związku z tym systemem hosta może być prawie każda dystrybucja systemu Linux. Aplikacja w kontenerze jest traktowana tak, jakby system operacyjny nie był kompletny, a posiadał tylko niezbędne pliki binarne, takie jak np. menedżer pakietów, serwer Apache-PHP i biblioteki potrzebne do uruchomienia aplikacji.

W przypadku wcześniejszej ilustracji pomarańczowa aplikacja mogła działać w dystrybucji Red Hat, a zielona w dystrybucji Debian, ale żadna z tych dystrybucji nie musiałaby być systemem hosta. Kolejną zaletą Dockera jest rozmiar obrazów. Obrazy nie muszą zawierać dużych elementów, takich jak jądro systemu operacyjnego lub cały system operacyjny. Dzięki temu są one bardzo małe i łatwe do przesyłania.

Instalacja Dockera

Instalator Dockera to pierwszy element tego ekosystemu, który należy uruchomić w swoim lokalnym środowisku, a także na serwerach. Oto lista środowisk, w których można zainstalować Dockera:

- Linux (różne dystrybucje);
- Apple macOS;
- Windows 10 Professional.

Ponadto instalator Dockera może zostać uruchomiony w chmurze takiej jak np. Amazon Web Services, Microsoft Azure i DigitalOcean. Docker integruje się w różny sposób z systemami operacyjnymi, dlatego dla każdego systemu przewidziano inny instalator. Podstawowym środowiskiem pracy Dockera jest Linux, a więc jeżeli korzystasz z tego systemu, jego uruchomienie nie będzie stanowiło żadnego problemu. Docker działa nieco inaczej w systemach macOS i Windows 10. Wynika to z tego, że korzysta on wewnętrznie z rozwiązań systemu Linux.

Przyjrzyjmy się procesowi instalacji Dockera na komputerze stacjonarnym pracującym pod kontrolą systemu Linux — Ubuntu 16.04, a następnie systemów macOS i Windows 10.

Instalacja w systemie Linux (Ubuntu 16.04)

Zgodnie z tym, co wcześniej pisaliśmy, instalacja Dockera w systemie Linux jest najprostsza. W celu zainstalowania Dockera uruchom poniższe polecenia w sesji Terminala:

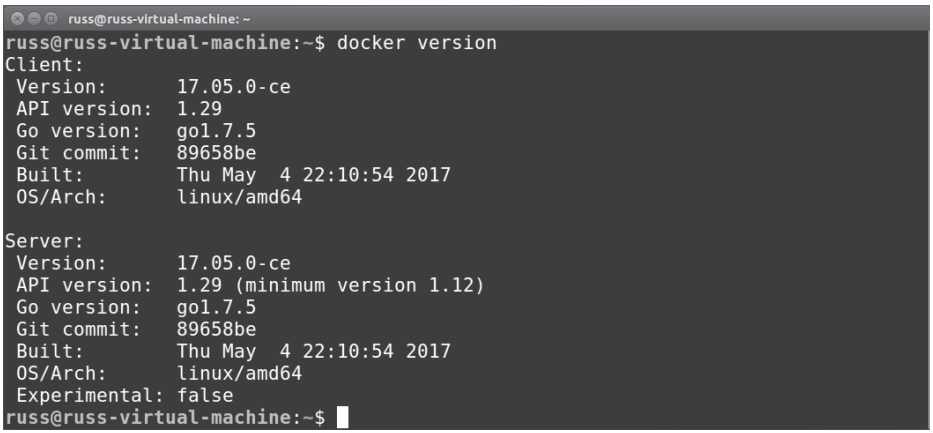
```
$ curl -sSL https://get.docker.com/ | sh
$ sudo systemctl start docker
```

Uruchomienie tych poleceń spowoduje pobranie, zainstalowanie i skonfigurowanie najnowszej wersji Dockera. Program zostanie pobrany bezpośrednio z serwera jego autorów. W momencie pisania tej książki najnowszą wersją Dockera instalowaną przez oficjalny skrypt jest 17.05.

W celu sprawdzenia poprawności instalacji i działania Dockera uruchom polecenie:

```
$ docker version
```

W Terminalu powinien pojawić się komunikat podobny do następującego:



W dalszej części książki będziemy korzystać z dwóch narzędzi pomocniczych, które są automatycznie instalowane wraz z Dockerem w systemach macOS i Windows. Zainstaluj je teraz, dzięki czemu nie będziesz tracić na to czasu później. Pierwszym z tych narzędzi jest Docker Machine. W celu zainstalowania go uruchom poniższe polecenia:

Aby sprawdzić, czy instalujesz najnowszą wersję narzędzia, wejdź w sekcję z wersjami znajdującą się na stronie projektu w repozytorium GitHub — *https://github.com/docker/machine/releases/*. W celu zainstalowania wersji innej niż v.0.13.0 po prostu umieść inny numer wersji w poniższym poleceniu.

```
$ curl -L https://github.com/docker/machine/releases/download/v0.13.0/
↪docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine &&
    chmod +x /tmp/docker-machine &&
    sudo cp /tmp/docker-machine /usr/local/bin/docker-machine
```

W celu pobrania i zainstalowania narzędzia Docker Compose uruchom poniższe polecenie. Tu również sprawdź, czy pobiera ono najnowszą wersję — wejdź na stronę *https://github.com/docker/compose/releases/*:

```
$ sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/
↪docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose &&
    chmod +x /usr/local/bin/docker-compose
```

Po poprawnym zainstalowaniu narzędzia można uruchomić następujące polecenia:

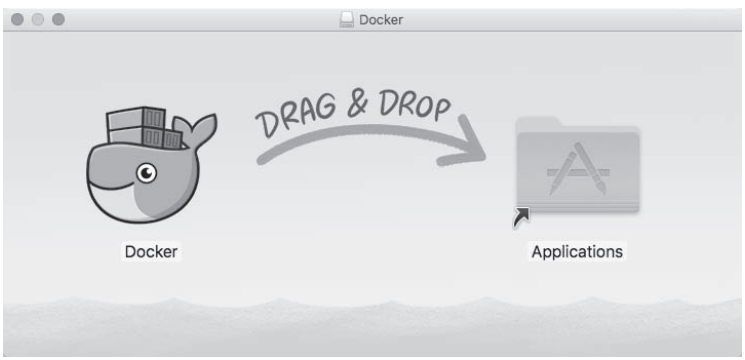
```
$ docker-compose version
$ docker-machine version
```

Instalacja w systemie macOS

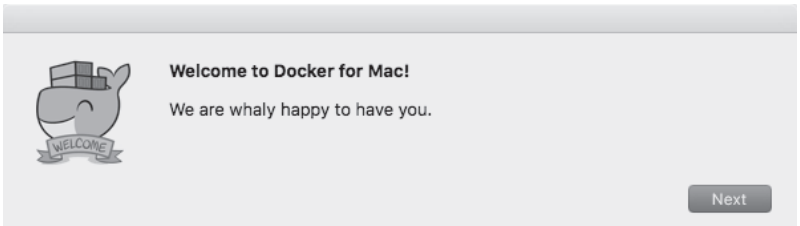
Docker w systemie macOS instalowany jest za pomocą graficznego instalatora.

Przed pobraniem instalatora upewnij się, że pracujesz w systemie Apple macOS Yosemite 10.10.3 lub nowszym. Jeżeli korzystasz ze starszej wersji tego systemu, to możesz również i w niej uruchomić Dockera — zajrzyj do sekcji „Starsze systemy operacyjne".

Instalator możesz pobrać z serwisu Docker Store (*https://store.docker.com/editions/community/docker-ce-desktop-mac/*). Kliknij przycisk *Get Docker* (pobierz Dockera). Spowoduje to pobranie pliku DMG. Kliknij go dwukrotnie w celu zamontowania obrazu i uruchomienia go. Na ekranie Twojego komputera powinno pojawić się następujące okno:



Po przeciągnięciu ikony Dockera do folderu aplikacji kliknij ją dwukrotnie. Zostaniesz zapytany, czy chcesz otworzyć pobraną aplikację. Kliknij *Tak*, co uruchomi instalator:

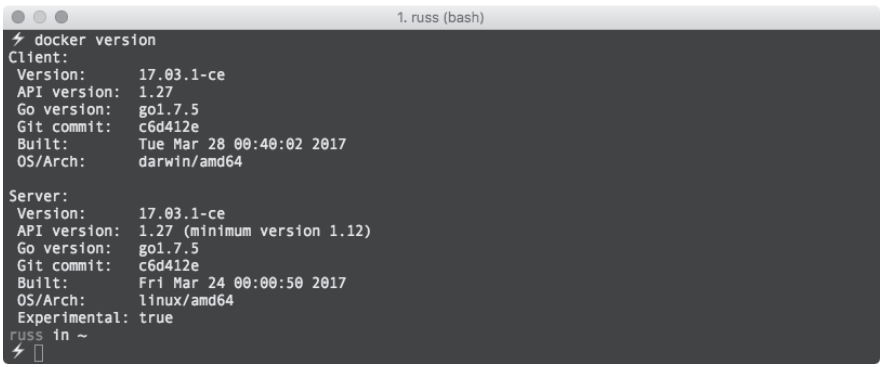


Kliknij przycisk *Next* (dalej) i wykonuj instrukcje wyświetlane na ekranie. Po zainstalowaniu i uruchomieniu Dockera jego ikona powinna pojawić się na belce widocznej w prawym górnym rogu ekranu. Kliknięcie tej ikony i wybranie opcji *About Docker* (o Dockerze) spowoduje wyświetlenie okna podobnego do poniższego:

Możesz również otworzyć okno Terminala i uruchomić w nim to samo polecenie, z którego korzystaliśmy w systemie Linux:

```
$ docker version
```

Powinno ono spowodować wyświetlenie informacji o Dockerze:



Oto polecenia, dzięki którym sprawdzisz wersje narzędzi Docker Compose i Docker Machine zainstalowanych wraz z silnikiem Docker Engine:

```
$ docker-compose version
$ docker-machine version
```

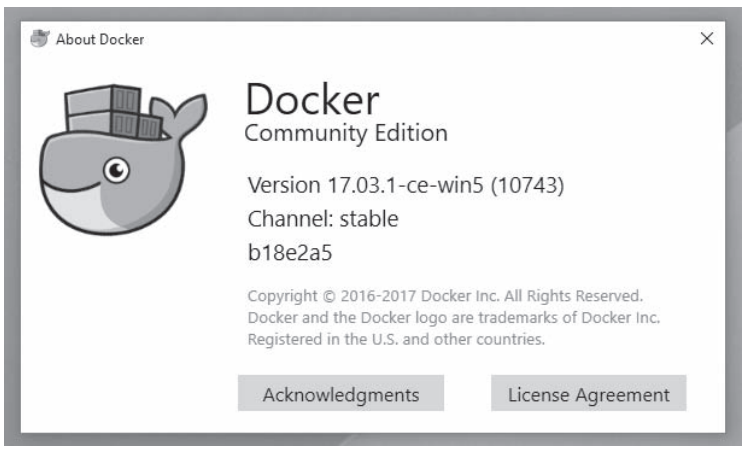
Instalacja w systemie Windows 10 Professional

W systemie Windows, podobnie jak w systemie macOS, Docker jest instalowany również za pomocą graficznego instalatora.

> Przed pobraniem instalatora upewnij się, że pracujesz w 64-bitowej wersji systemu Microsoft Windows 10 Professional lub Enterprise. Jeżeli korzystasz ze starszej wersji tego systemu lub nieobsługiwanej wersji Windows 10, zajrzyj do sekcji „Starsze systemy operacyjne".
>
> Docker wymaga określonej wersji systemu Windows, ponieważ korzysta z Hyper-V — hipernadzorcy wbudowanego w system Windows, umożliwiającego uruchamianie aplikacji o architekturze x86-64. Mogą to być aplikacje systemu Windows 10 Professional lub Windows Server. Wspomniany hipernadzorca wchodzi nawet w skład systemu operacyjnego konsoli XBox One.

Instalator Dockera dla systemu Windows możesz pobrać z serwisu Docker Store *https://store.docker.com/editions/community/docker-ce-desktop-windows/*. Kliknij przycisk *Get Docker* (pobierz Dockera) w celu pobrania instalatora. Po uruchomieniu instalatora Docker zostanie automatycznie rozpakowany i zainstalowany. Opcja uruchomienia hipernadzorcy pojawi się przy pierwszym uruchomieniu Dockera.

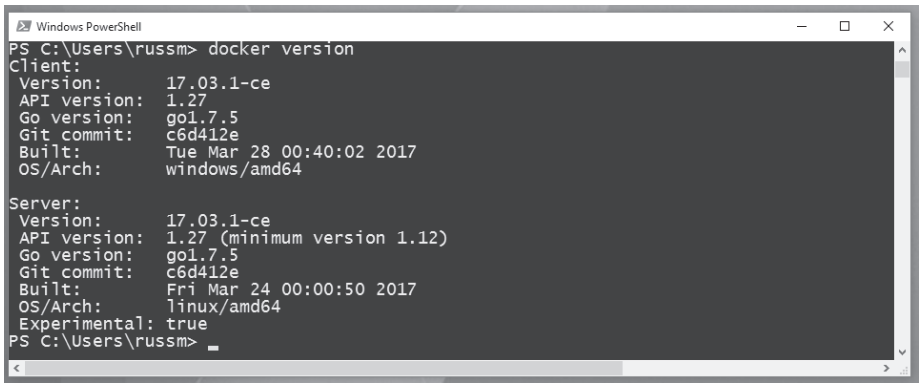
Po zakończeniu procesu instalacji ikona Dockera pojawi się w zasobniku systemowym wyświetlanym w prawym dolnym rogu ekranu. Kliknij ją i wybierz opcję *About Docker*, co spowoduje wyświetlenie okna z informacjami o zainstalowanej wersji Dockera:



Otwórz okno *PowerShell* i uruchom w nim polecenie:

```
$ docker version
```

W oknie pojawią się informacje podobne do tych, które były wyświetlane w wersjach przeznaczonych dla systemów Mac i Linux:



W systemie również możesz uruchomić następujące polecenia:

```
$ docker-compose version
$ docker-machine version
```

Starsze systemy operacyjne

Jeżeli korzystasz ze starszej wersji systemu macOS lub Windows, to musisz skorzystać z narzędzia Docker Toolbox. Być może zauważyłeś, że po uruchomieniu polecenia:

```
$ docker version
```

w przypadku zaprezentowanych trzech instalacji wyświetlone zostały informacje o dwóch wersjach: klienta i serwera. W przypadku instalacji w systemie Linux komunikat zawierał informacje o tym, że architektura klienta i serwera to Linux; w przypadku wersji zainstalowanej w systemie macOS komunikat informował o tym, że klient działa na architekturze Darwin, czyli uniksowym jądrze firmy Apple, a w systemie Windows klient działa na architekturze systemu Windows, ale wszystkie serwery działają według komunikatu na architekturze systemu Linux. Jak to możliwe?

Wynika to z tego, że wersje Dockera pracujące w systemach Windows i macOS pobierają i uruchamiają w tle maszynę wirtualną, w której uruchomiony jest lekki system operacyjny Alpine Linux. Maszyna ta korzysta z własnych bibliotek Dockera, które łączą się z wbudowanym hipernadzorcą wybranego środowiska. W przypadku systemu macOS jest to wbudowany moduł Hypervisor Framework (*https://developer.apple.com/documentation/hypervisor*), a w systemie Windows jest to Hyper-V (*https://www.microsoft.com/en-gb/cloud-platform/server-virtualization*).

Istnieją wersje Dockera przeznaczone dla starszych wersji systemów macOS i Windows. Nie korzystają one z wbudowanych hipernadzorców. Zamiast nich używają programu VirtualBox w charakterze hipernadzorcy — lokalny klient łączy się z uruchomionym w tym programie serwerem Linux.

> VirtualBox to otwarte narzędzie wirtualizacyjne przeznaczone dla architektur x86 i AMD64/Intel64 opracowane przez firmę Oracle. Działa ono w hostach Windows, Linux, Macintosh i Solaris. Obsługuje wiele wirtualizowanych systemów operacyjnych z rodzin Linux, Unix i Windows. Więcej informacji na temat programu VirtualBox znajdziesz na stronie *https://www.virtualbox.org*.

Więcej informacji na temat projektu Docker Toolbox znajdziesz na stronie *https://www.docker.com/get-docker*, z której możesz również pobrać programy instalacyjne systemów macOS i Windows.

> W tej książce zakładamy, że zainstalowałeś najnowszą wersję Dockera w systemie Linux lub korzystasz z Dockera w wersji dla systemu macOS lub Windows. Co prawda instalacje wykonane przy użyciu narzędzia Docker Toolbox powinny obsługiwać polecenia opisane w tej książce, ale pracując w takiej instalacji, możesz spotkać się z problemami wynikającymi z braku uprawnień do plików podczas montowania w kontenerze danych z lokalnej maszyny.

Klient Dockera w wierszu poleceń

Po zainstalowaniu Dockera czas przypomnieć sobie jego najważniejsze polecenia. Zaczniemy od tych, które są używane najczęściej, przejdziemy do poleceń, które są używane podczas zarządzania obrazami, a na koniec zajmiemy się poleceniami dotyczącymi kontenerów.

> Docker niedawno zmienił strukturę klienta obsługiwanego z poziomu wiersza poleceń. Polecenia zostały pogrupowane w sposób bardziej logiczny. Zabieg ten wynikał z szybkiego rozwijania się platformy i krzyżowania poleceń. W tej książce będziemy korzystać z nowej struktury. Więcej informacji na temat zmian wprowadzonych w kliencie obsługiwanym za pomocą wiersza poleceń znajdziesz w artykule *https://blog.docker.com/2017/01/whats-new-in-docker-1-13/*.

Zaczniemy od najważniejszego polecenia — tego wyświetlającego pomoc. W celu uruchomienia pomocy uruchom polecenie:

```
$ docker help
```

Na ekranie pojawi się lista wszystkich dostępnych poleceń Dockera wraz z krótkimi opisami ich funkcji. W celu uzyskania bardziej szczegółowych informacji dotyczących wybranego polecenia wpisz:

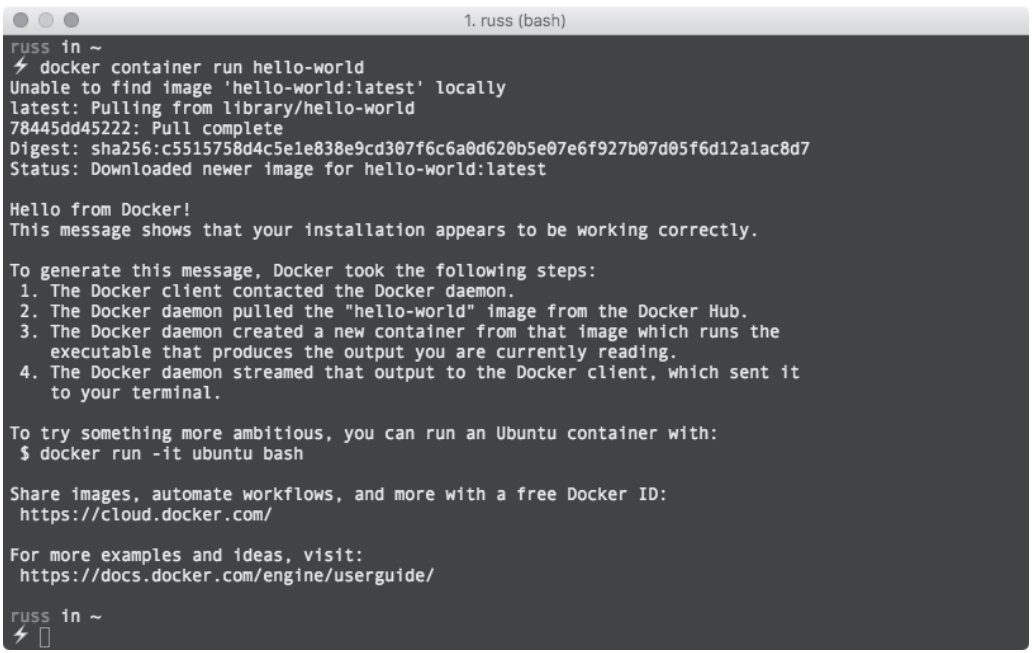
```
$ docker <POLECENIE> --help
```

Teraz możemy przystąpić do uruchomienia kontenera `hello-world`. Zróbmy to za pomocą polecenia (w zależności od konfiguracji systemu Linux może istnieć konieczność poprzedzenia poleceń instrukcją `sudo`):

```
$ docker container run hello-world
```

Nieważne, na jakim hoście uruchomiono Dockera. W systemach Linux, macOS i Windows zostaną wykonane te same operacje — Docker pobierze obraz kontenera `hello-world`, uruchomi go, a następnie zatrzyma po wykonaniu znajdującego się w nim kodu.

W oknie Terminala powinny zostać wyświetlone następujące komunikaty:

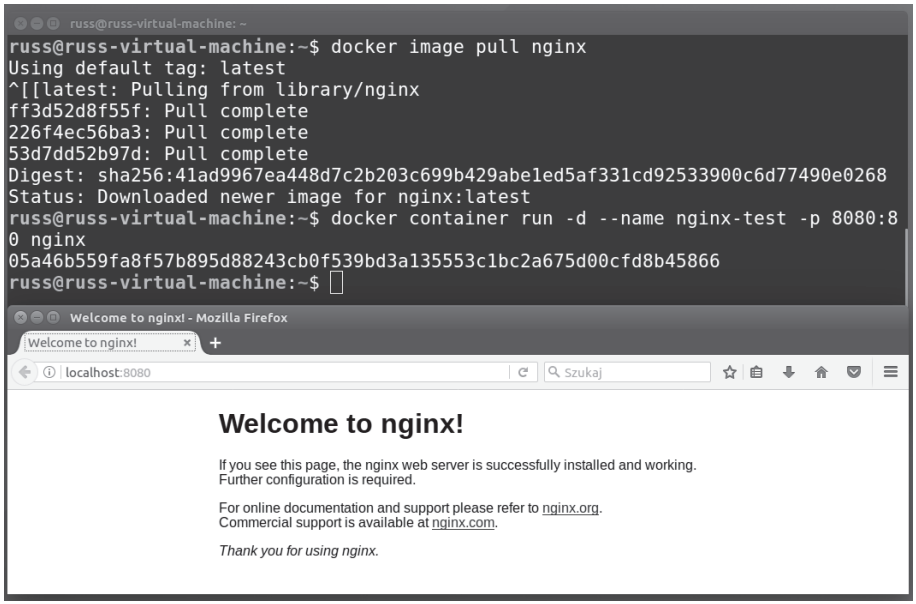


Zróbmy coś ciekawszego. Spróbujmy pobrać i uruchomić kontener NGINX. W tym celu należy uruchomić następujące polecenia:

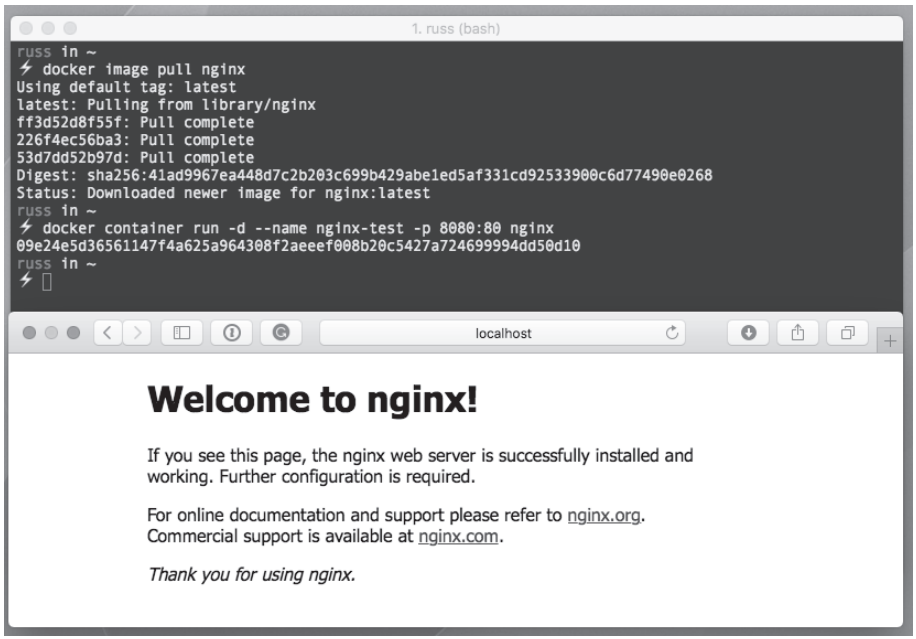
```
$ docker image pull nginx
$ docker container run -d --name nginx-test -p 8080:80 nginx
```

Pierwsze polecenie pobiera obraz kontenera NGINX, a drugie uruchamia go w tle — tworzy kontener `ngnix-test` na bazie pobranego obrazu `ngnix`. Drugie polecenie mapuje również port 8080 hosta do portu 80 kontenera, dzięki czemu można uzyskać do niego dostęp lokalnie za pomocą przeglądarki internetowej (pod adresem *http://localhost:8080/*).

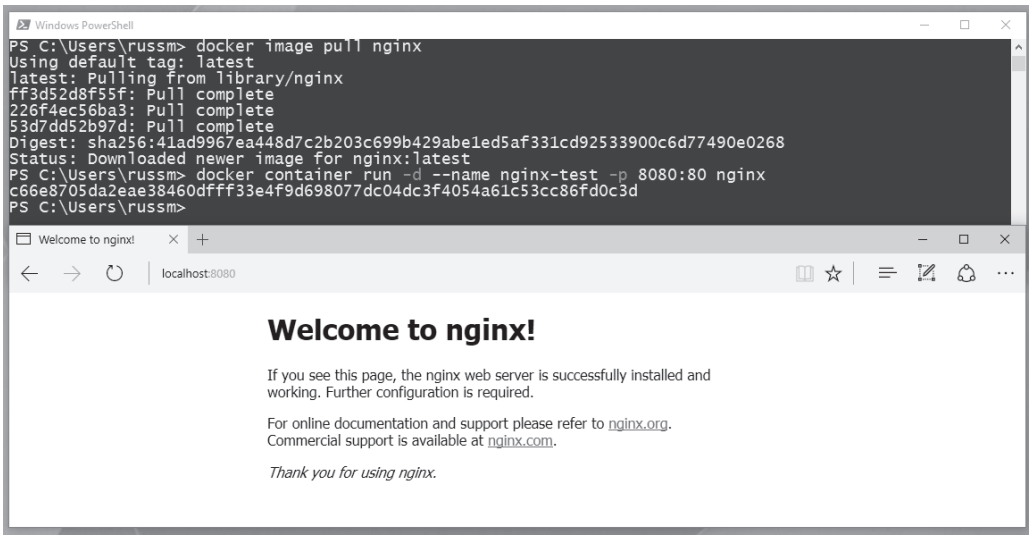
Polecenia te zadziałają tak samo we wszystkich trzech systemach operacyjnych (o czym przekonasz się, patrząc na trzy kolejne rysunki). Oto zrzut z systemu Linux:



Oto zrzut z systemu macOS:



A to zrzut z systemu Windows:



W kolejnych trzech rozdziałach poszerzysz swoją wiedzę dotyczącą klienta obsługiwanego za pomocą wiersza poleceń. Zatrzymaj i usuń kontener `nginx-test` za pomocą poleceń:

```
$ docker container stop nginx-test
$ docker container rm nginx-test
```

Ekosystem Dockera

Docker poza dołączonymi do niego narzędziami obsługuje również wiele dodatkowych narzędzi. Część z nich opiszemy w kolejnych rozdziałach. Na koniec pierwszego rozdziału chcieliśmy stworzyć zarys narzędzi, z których będziemy korzystać. Najważniejszym z nich jest **Docker Engine**.

To jądro Dockera. Korzystają z niego wszystkie pozostałe opisane przez nas narzędzia. Korzystaliśmy z niego w sekcjach związanych z instalacją Dockera. Obecnie utrzymywane są dwie wersje silnika Docker Engine: **Docker Enterprise Edition** (**Docker EE**) i **Docker Community Edition** (**CE**). W tej książce korzystamy z Dockera CE. Niedawno cykl wydawania nowych wersji silnika Docker Engine został zmieniony, dzięki czemu jest bardziej przewidywalny.

Silniki Docker CE i EE mają stabilne wersje, które są aktualizowane co kwartał. Jak zapewne zauważyłeś, w systemie macOS i Windows zainstalowaliśmy Dockera w wersji CE v17.03.x. Istnieje jeszcze wersja Edge, w której nowe funkcje są wprowadzane co miesiąc, ale lepiej nie korzystać z niej w środowiskach produkcyjnych. W systemie Linux zainstalowaliśmy Dockera w wersji Edge CE 17.05.x. Oto inne elementy ekosystemu Dockera:

- **Docker Compose** to narzędzie pozwalające na tworzenie i udostępnianie zespołów kontenerów. Więcej informacji na ten temat znajdziesz w rozdziale 6. „Docker Compose".
- **Docker Machine** to narzędzie przeznaczone do uruchamiania hostów Dockera na różnych platformach. Więcej informacji na ten temat znajdziesz w rozdziale 5. „Docker Machine".
- **Docker Hub** to repozytorium przeznaczone do przechowywania obrazów Dockera. Zagadnienia z nim związane zostaną poruszone w trzech kolejnych rozdziałach.
- **Docker Store** to serwis przeznaczony do pobierania oficjalnych obrazów i wtyczek Dockera, a także innych licencjonowanych produktów. Zagadnienia związane z tym tematem również zostaną poruszone w trzech kolejnych rozdziałach.
- **Docker Swarm** to narzędzie przeznaczone do orkiestracji mogące pracować z wieloma hostami. Więcej informacji na ten temat znajdziesz w rozdziale 7. „Docker Swarm".
- **Docker dla systemu macOS**. Zagadnienie to opisaliśmy w tym rozdziale.
- **Docker Cloud** — platforma **CaaS** (**kontener jako usługa**) opisana jest szczegółowo w rozdziale 10. „Usługa Docker Cloud".
- **Docker dla systemu Windows**. Zagadnienie to opisaliśmy w tym rozdziale.
- **Docker dla usług Amazon Web Services**. Najlepsze praktyki instalacji narzędzia Docker Swarm w kontekście obsługi usług AWS opiszemy w rozdziale 10. „Usługa Docker Cloud".
- **Docker dla Azure**. Najlepsze praktyki instalacji narzędzia Docker Swarm w kontekście obsługi usługi Azure opiszemy w rozdziale 10. „Usługa Docker Cloud".

W kolejnych rozdziałach przyjrzymy się również wybranym usługom niezależnych twórców.

Podsumowanie

W tym rozdziale przedstawiliśmy podstawowe zagadnienia, które są niezbędne podczas lektury kolejnych rozdziałów. Dowiedziałeś się, czym jest i jak działa Docker. Opisaliśmy przebieg instalacji Dockera w różnych systemach operacyjnych i obsługę podstawowych funkcji Dockera z poziomu wiersza poleceń. Pamiętaj o konieczności sprawdzenia wymagań przed uruchomieniem instalatora — korzystaj z instalatora właściwego dla swojego systemu operacyjnego.

W dalszej części tego rozdziału opisaliśmy podstawowe polecenia umożliwiające rozpoczęcie pracy z Dockerem. W kolejnych rozdziałach opiszemy wszystkie polecenia służące do zarządzania kontenerami. Przedstawimy działanie poszczególnych poleceń, a także ich zastosowania. Pod koniec tego rozdziału wymieniliśmy narzędzia wchodzące w skład ekosystemu Dockera.

W kolejnych rozdziałach przyjrzymy się procesowi tworzenia bazowych kontenerów, opiszemy zagadnienia związane z plikami Dockerfile, przechowywaniem obrazów, a także korzystaniem ze zmiennych środowiskowych i wolumenów Dockera.

Skorowidz

E

F

G

H

I

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

V

W

Z

Ź

Ż

Docker: od teraz aplikacja działa zawsze i wszędzie!

Od czasu premiery w 2013 roku Docker cieszy się rosnącym zainteresowaniem. Liczba deweloperów korzystających z tej platformy rośnie lawinowo. Docker zmienił sposób projektowania i wdrażania aplikacji, włączając w to aplikacje sieciowe. Ma duże możliwości i łączy w sobie prostotę wdrażania aplikacji z prostotą administrowania. Pozwala na rozwiązywanie problemów, z którymi borykają się zespoły programistów i administratorów wdrażających nowe systemy. Jednym słowem, jest to niezwykle użyteczne narzędzie i warto maksymalnie wykorzystać jego potencjał!

Niniejsza książka jest praktycznym podręcznikiem, dzięki któremu szybko zaczniesz efektywnie korzystać z Dockera. Prędko też zauważysz zupełnie nowe możliwości pracy nad oprogramowaniem. Zapoznasz się z podstawowymi koncepcjami związanymi z Dockerem i z takimi zagadnieniami jak budowanie, zarządzanie i przechowywanie obrazów. Dowiesz się, kiedy i w jaki sposób warto rozszerzyć Dockera oraz jak zintegrować go z różnymi platformami i narzędziami. Nauczysz się pracować z kontenerami za pomocą narzędzi Docker Machine, Docker Swarm i Docker Compose. Zapoznasz się również z problematyką bezpieczeństwa tworzonych systemów.

W tej książce między innymi:

- zwięzłe podstawy Dockera oraz zasady pracy z obrazami i kontenerami Dockera
- przechowywanie i dystrybucja obrazów
- praca z narzędziami Portainer i Rancher oraz z usługą Docker Cloud
- zabezpieczanie platformy i zarządzanie przepływem zadań
- ulepszanie aplikacji działającej w kontenerze Dockera

Russ McKendrick — jest architektem oprogramowania. Z branżą IT jest związany od ponad ćwierćwiecza. Korzysta niemal wyłącznie z rozwiązań open source, a Docker jest jego ulubionym narzędziem. Pasjonuje się muzyką na płytach winylowych.
Scott Gallagher — od dzieciństwa fascynuje się programowaniem. Zajmował się takimi technologiami jak serwerowe rozwiązania firm Novell, Microsoft i Red Hat. Obecnie interesuje się środowiskami systemu Linux, a w pracy skupia się na Dockerze i technologiach chmury.

ISBN 978-83-283-4308-5
9 788328 343085
Cena: 59,00 zł
Packt