

Russ McKendrick, Scott Gallagher

Docker

Programowanie
aplikacji dla
zaawansowanych

Wydanie II



Packt

Tytuł oryginału: Mastering Docker - Second Edition

Tłumaczenie: Konrad Matuk

ISBN: 978-83-283-4309-2

Copyright © Packt Publishing 2017. First published in the English language
under the title 'Mastering Docker - Second Edition – (9781787280243)'

Polish edition copyright © 2018 by Helion SA
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any
form or by any means, electronic or mechanical, including photocopying, recording
or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości
lub fragmentu niniejszej publikacji w jakiekolwiek postaci jest zabronione.
Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopowanie
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 właścicielu.

Autor oraz Wydawnictwo HELION 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 Wydawnictwo HELION
nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe
z wykorzystania informacji zawartych w książce.

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_ebook
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

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

- [Poleć książkę na Facebook.com](#)
- [Kup w wersji papierowej](#)
- [Oceń książkę](#)

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

Spis treści

| | |
|--|-----------|
| O autorach | 9 |
| O recenzentach | 11 |
| Wstęp | 13 |
| | |
| Rozdział 1. Docker — wprowadzenie | 17 |
| Czym jest Docker? | 17 |
| Programiści | 18 |
| Administratorzy | 19 |
| Firmy | 21 |
| Różnice pomiędzy dedykowanymi hostami, maszynami wirtualnymi i Dockerem | 22 |
| Instalacja Dockera | 23 |
| Instalacja w systemie Linux (Ubuntu 16.04) | 24 |
| Instalacja w systemie macOS | 25 |
| Instalacja w systemie Windows 10 Professional | 27 |
| Starsze systemy operacyjne | 28 |
| Klient Dockera w wierszu poleceń | 29 |
| Ekosystem Dockera | 32 |
| Podsumowanie | 33 |
| | |
| Rozdział 2. Tworzenie obrazów kontenerów | 35 |
| Plik Dockerfile — wprowadzenie | 35 |
| Szczegółowa analiza pliku Dockerfile | 37 |
| Dobre praktyki pracy z plikami Dockerfile | 41 |
| Budowanie obrazów Dockera | 42 |
| Polecenie docker build | 42 |
| Korzystanie z utworzonego wcześniej kontenera | 44 |
| Budowanie od podstaw | 48 |
| Zmienne środowiskowe | 50 |
| Umieszczanie zmiennych środowiskowych w pliku Dockerfile | 50 |
| Czas wszystko połączyć ze sobą | 50 |
| Podsumowanie | 56 |

| | |
|---|------------|
| Rozdział 3. Przechowywanie obrazów i ich dystrybucja | 57 |
| Repozytorium Docker Hub | 57 |
| Panel główny | 58 |
| Przycisk Explore | 59 |
| Przycisk Organizations | 60 |
| Menu Create | 60 |
| Profil i ustawienia | 61 |
| Strona Stars | 62 |
| Automatyzacja budowy obrazu | 63 |
| Ładowanie własnych obrazów | 68 |
| Serwis Docker Store | 70 |
| Rejestr Docker Registry | 71 |
| Docker Registry — informacje ogólne | 71 |
| Wdrażanie własnego rejestru | 72 |
| Rejestr Docker Trusted Registry | 74 |
| Niezależne rejestyry | 75 |
| Quay | 75 |
| Rejestr Amazon EC2 Container Registry | 78 |
| Microbadger | 79 |
| Podsumowanie | 82 |
| Rozdział 4. Zarządzanie kontenerami | 83 |
| Polecenia służące do obsługi kontenerów Dockera | 84 |
| Podstawy | 84 |
| Komunikowanie się z kontenerami | 88 |
| Dzienniki i informacje o procesach | 91 |
| Ograniczenia zasobów | 93 |
| Stany kontenerów i pozostałe polecenia | 95 |
| Usuwanie kontenerów | 97 |
| Obsługa sieci i wolumenów | 100 |
| Sieć | 100 |
| Wolumeny Dockera | 107 |
| Podsumowanie | 112 |
| Rozdział 5. Docker Machine | 113 |
| Wprowadzenie do narzędzia Docker Machine | 113 |
| Docker Machine i wdrażanie lokalnych hostów Dockera | 114 |
| Uruchamianie hostów Dockera w chmurze | 119 |
| DigitalOcean | 119 |
| Amazon Web Services | 121 |
| Więcej o sieciowych możliwościach Dockera | 124 |
| Stosowanie innych bazowych systemów operacyjnych | 128 |
| System CoreOS w chmurze DigitalOcean | 128 |
| System RancherOS w maszynie VirtualBox | 130 |
| Podsumowanie | 131 |

| | |
|--|------------|
| Rozdział 6. Docker Compose | 133 |
| Wprowadzenie do Docker Compose | 133 |
| Uruchamianie aplikacji za pomocą narzędzia Docker Compose | 135 |
| Plik YAML narzędzia Docker Compose | 137 |
| Plik YAML aplikacji Mobycounter | 137 |
| Aplikacja do głosowania | 139 |
| Polecenia Docker Compose | 146 |
| Up i PS | 147 |
| Config | 148 |
| Pull, build i create | 148 |
| Start, stop, restart, pause i unpause | 149 |
| Top, logs i events | 149 |
| Exec i run | 151 |
| Scale | 152 |
| Kill, rm i down | 153 |
| Podsumowanie | 154 |
| Rozdział 7. Docker Swarm | 155 |
| Docker Swarm — instalacja | 156 |
| Role Docker Swarm | 156 |
| Menedżer Swarm | 157 |
| Wykonawca Swarm | 157 |
| Korzystanie z trybu Docker Swarm | 158 |
| Tworzenie klastra | 158 |
| Dołączanie wykonawców | 160 |
| Listy węzłów | 161 |
| Zarządzanie klastrem | 161 |
| Promowanie hosta roboczego | 164 |
| Degradowanie węzła menedżera | 165 |
| Drenaż węzła | 166 |
| Usługi i stosy Docker Swarm | 168 |
| Usługi | 168 |
| Stosy | 171 |
| Kasowanie klastra Swarm | 173 |
| Równoważenie obciążeń, nakładki i tworzenie harmonogramów | 174 |
| Równoważenie obciążień wejściowych | 174 |
| Nakładki sieciowe | 175 |
| Tworzenie harmonogramu | 176 |
| Podsumowanie | 176 |
| Rozdział 8. Portainer | 177 |
| Historia prac nad narzędziem Portainer | 177 |
| Uruchamianie narzędzia Portainer | 178 |
| Korzystanie z narzędzia Portainer | 180 |
| Panel główny | 181 |
| Szablony aplikacji | 181 |

| | |
|---|------------|
| Kontenery | 183 |
| Obrady | 187 |
| Sieci i wolumeny | 190 |
| Zdarzenia | 190 |
| Docker | 191 |
| Portainer i Docker Swarm | 191 |
| Tworzenie klastra | 192 |
| Usługa Portainer | 193 |
| Różnice związane z pracą w klastrze | 194 |
| Podsumowanie | 199 |
| Rozdział 9. Rancher | 201 |
| Instalacja i konfiguracja uwierzytelniania | 201 |
| Instalacja | 202 |
| Konfiguracja uwierzytelniania | 204 |
| Tworzenie stada | 207 |
| Uruchamianie stosów | 209 |
| Stosy definiowane przez użytkownika | 210 |
| Podgląd właściwości kontenerów | 216 |
| Katalog | 217 |
| Usuwanie stada | 217 |
| Inne klastry | 218 |
| Podsumowanie | 221 |
| Rozdział 10. Usługa Docker Cloud | 223 |
| Zakładanie konta | 224 |
| Łączenie kont | 225 |
| DigitalOcean | 226 |
| Amazon Web Services | 227 |
| Uruchamianie węzłów | 231 |
| Uruchamianie stosu | 234 |
| Tryb Swarm | 238 |
| Docker dla Amazon Web Services | 239 |
| Podsumowanie | 244 |
| Rozdział 11. Bezpieczeństwo platformy Docker | 245 |
| Bezpieczeństwo kontenerów | 245 |
| Zalety | 246 |
| Host Dockera | 246 |
| Zaufane źródła obrazów | 247 |
| Polecenia Dockera | 247 |
| Polecenie run | 247 |
| Polecenie diff | 249 |
| Dobre praktyki | 250 |
| Dobre praktyki pracy w Dockerze | 250 |
| Zalecenia organizacji Center for Internet Security | 251 |

| | |
|--|------------|
| Aplikacja Docker Bench Security | 252 |
| Uruchamianie narzędzia w systemach macOS i Windows | 253 |
| Uruchamianie narzędzia w systemie Linux Ubuntu | 253 |
| Analiza zwracanych informacji | 255 |
| Docker Bench — podsumowanie | 260 |
| Skanowanie zabezpieczeń Dockera | 260 |
| Niezależne usługi poprawiające bezpieczeństwo | 262 |
| Quay | 262 |
| Clair | 263 |
| Podsumowanie | 264 |
| <hr/> Rozdział 12. Przepływ zadań w platformie Docker | 265 |
| Docker i prace programistyczne | 265 |
| Monitorowanie | 277 |
| Rozszerzanie na zewnętrzne platformy | 286 |
| Instalator Tectonic | 286 |
| Platforma Heroku | 289 |
| Usługa Amazon Elastic Container Service | 290 |
| Jak wygląda środowisko produkcyjne? | 291 |
| Hosty Dockera | 291 |
| Obsługa klastrów | 292 |
| Rejestry obrazów | 293 |
| Podsumowanie | 294 |
| <hr/> Rozdział 13. Dalsze kroki z Dockerem | 295 |
| Wykrywanie usług | 295 |
| Consul | 295 |
| Narzędzie etcd | 304 |
| Interfejs Docker API | 304 |
| Projekt Moby Project | 306 |
| Własny wkład w rozwój Dockera | 308 |
| Rozwój kodu | 308 |
| Pomoc innym | 309 |
| Inny wkład w rozwój Dockera | 310 |
| Podsumowanie | 310 |
| <hr/> Skorowidz | 311 |

Zespół wydania oryginalnego

Authors

Russ Kendrick
Scott Gallagher

Reviewers

Srdjan Grubor
Christopher Janietz

Acquisition Editor

Prateek Bharadwaj

Content Development Editor

Sharon Raj

Technical Editor

Prashant Chaudhari

Copy Editor

Madhusudan Uchil

Project Coordinator

Virginia Dias

Proofreader

Safis Editing

Indexer

Mariammal Chettiyar

Graphics

Kirk D'Penha

Production Coordinator

Aparna Bhagat

O autorach

Russ Kendrick to architekt oprogramowania od ponad 25 lat pracujący w branży IT. W swojej karierze zajmował się różnymi rzeczami — administrował całą infrastrukturą, pracował na pierwszej linii frontu, a także zarządzał projektami wewnętrznymi i zewnętrznymi w małych firmach oraz w wielkich korporacjach.

Russ pracuje praktycznie wyłącznie w systemie Linux i korzysta z otwartych systemów i narzędzi, obsługując dedykowane serwery, a także maszyny wirtualne wchodzące w skład publicznych i prywatnych chmur firmy Node4 Limited, w której jest kierownikiem zespołu tworzącego rozwiązania w oparciu o otwarte oprogramowanie.

W międzyczasie Russ napisał trzy wydane przez Packt książki o Dockerze: *Monitoring Docker*, *Extending Docker* i *Docker Bootcamp*. Ponadto jest współautorem wydanej przez The New Stack książki *Monitoring and Management With Docker and Containers*.

Poza tym Russ zbyt często kupuje płyty winylowe.

*Cheialbym podziękować za wsparcie ze strony rodziny i przyjaciół, a także za ich zrozumienie, gdy siedziałem przed komputerem, pracując nad tą publikacją.
Ponadto dziękuję kolegom z firmy Node4 i wszystkim naszym klientom za wsparcie przez cały okres pracy nad książką.*

Scott Gallagher fascynuje się nowymi technologiami od szkoły podstawowej, gdy zagrał w grę *Oregon Trail*. Zamiłowanie do komputerów rozwijał również w dalszych latach nauki, pracując na komputerach Apple IIe. W szkole średniej nauczył się składać komputery i programować w języku BASIC. W koledżu zajmował się głównie technologiami serwerowymi firm takich jak Novell, Microsoft i Red Hat. Po zakończeniu nauki pracował jako programista w firmie Novell, ale interesował się również innymi technologiami. W dalszej kolejności zarządzał środowiskami firmy Microsoft, a na koniec najbardziej zainteresował się środowiskami systemu Linux. Teraz jego praca skupia się na Dockerze i technologiach chmury.

Chciałbym podziękować mojej rodzinie za wsparcie podczas pracy nad tą książką, a także za to, że pomagali mi przez całe życie. Szczególne podziękowania kieruję do mojej żony, która jest moją bratnią duszą, miłością mojego życia, najważniejszą osobą, a także powodem tego, że codziennie staram się robić wszystko jak najlepiej. Dziękuję również moim dzieciom, które są najbardziej niezwykłymi osobami na świecie. Codzienne podziwianie tego, jak rosną, jest dla mnie prawdziwym błogosławieństwem. Na koniec chciałbym podziękować moim rodzicom — dzięki nim stałem się tym, kim dzisiaj jestem.

O recenzentach

Srdjan Grubor jest inżynierem oprogramowania mającym doświadczenie w pracy nad dużymi i małymi projektami. Tworzył jednowierszowe skrypty, ale opracował również dwie dystrybucje systemu Linux. Obecnie zajmuje się między innymi tworzeniem platformy PaaS dla systemu Endless OS. Platforma ta ma być oparta na Dockerze, co pozwoli osobom pracującym w różnych miejscach na wykorzystanie wszystkich możliwości internetu poprzez agregację wiedzy i asynchroniczne buforowanie. Srdjan lubi rozbierać różne urządzenia, aby zobaczyć, jak działają, a także rozwiązywać skomplikowane problemy. Ponadto uważa, że w technologii zawsze jest miejsce dla filantropii.

Christopher Janietz jest tak zwanym DevOpem — wykonuje zadania programisty i administratora. Obecnie pracuje w MediaMarktSaturn, będącym największym sprzedawcą detalicznym sprzętu elektronicznego w Europie. Ponadto tworzy treści na potrzeby niemieckiego serwisu LinkedIn — jest autorem filmów instruktażowych związanych z programowaniem, a także automatyzacją infrastruktury. Christopher w wolnym czasie uwielbia przeglądać prace naukowe i codziennie powiększać swoją wiedzę na temat nowych technologii.

*Chciałbym podziękować moim kolegom i przyjaciołom — mistrzom Dockera
— Andreasowi Gero i Emanuelowi Kechterowi.*

Wstęp

Docker zmienił sposób projektowania i wdrażania nowoczesnych aplikacji — stał się głównym czynnikiem innowacji wykraczających poza zagadnienia związane z zarządzaniem systemami. Technologia ta wpływa również między innymi na tworzenie aplikacji sieciowych. Czy jesteś pewny tego, że nadążasz za wszystkimi nowinkami związanymi z Dockerem? Czy korzystasz z jego pełnego potencjału?

Niniejsza książka pokaże Ci nie tylko, jak korzystać z Dockera w sposób bardziej efektywny, ale także jej lektura sprawi, że zauważysz zupełnie nowe możliwości pracy nad oprogramowaniem.

Przed zagłębieniem się w trudniejsze tematy związane z bezpieczeństwem opiszemy podstawowe zagadnienia, takie jak budowanie obrazów, zarządzanie nimi i przechowywanie ich, a także przedstawimy najlepsze praktyki pracy z Dockerem. Podczas lektury dowiesz się o wszystkim, co dotyczy rozszerzania i integrowania Dockera za pomocą nowych i innowacyjnych rozwiązań. Nauczysz się pracować ze swoimi kontenerami za pomocą narzędzi Docker Swarm i Docker Compose.

Dzięki lekturze tej książki poznasz dokładnie możliwości Dockera, a także dowiesz się, jak łączyć go z różnymi platformami i narzędziami:

- Poznasz podstawowe elementy Dockera i związane z nimi koncepcje.
- Zabezpieczysz kontenery i pliki za pomocą zabezpieczeń obsługiwanych przez Dockera.
- Rozszerzysz możliwości Dockera i rozbudujesz architekturę za pomocą niezależnych narzędzi orkiestracji, usług i wtyczek.
- Podczas pracy nad wysoce skalowalnymi aplikacjami skorzystasz z technologii wirtualizacji kontenerów systemu Linux.

Zawartość tej książki

Rozdział 1. „Docker — wprowadzenie” przedstawia podstawowe zagadnienia związane z Dockerem. Dzięki lekturze tego rozdziału przypomnisz sobie pewne rzeczy, o których możesz zapomnieć, i ułatwisz sobie przyswajanie wiedzy zawartej w kolejnych rozdziałach.

Rozdział 2. „Tworzenie obrazów kontenerów” opisuje proces tworzenia własnych obrazów bazowych. Dowiesz się, jak tworzy się aplikacje umieszczone w obrazie, a także poznasz ważne zaawansowane zagadnienia związane z samodzielnym tworzeniem obrazów.

Rozdział 3. „Przechowywanie obrazów i ich dystrybucja” dotyczy miejsc przechowywania obrazów, takich jak Docker Hub i lokalne rejestrury. Opiszemy również niezależne rejestrury, takie jak Quay i Amazon EC2 Container Registry.

Rozdział 4. „Zarządzanie kontenerami” przedstawia różne techniki zarządzania kontenerami. W tym rozdziale skupimy się głównie na wierszu polecień.

Rozdział 5. „Docker Machine” opisuje obsługę narzędzia Docker Machine, tworzenie nowych hostów Dockera, przechowywanie kontenerów lokalnie, a także w wirtualnych magazynach i chmurach, takich jak np. Amazon Web Services i DigitalOcean.

Rozdział 6. „Docker Compose” przedstawia prostotę przekazywania środowisk Dockera programistom i konfigurowania ich zgodnie z wymaganiami systemów produkcyjnych.

Rozdział 7. „Docker Swarm” opisuje rozwiązanie umożliwiające łączenie ze sobą kontenerów znajdujących się na różnych serwerach. Zastosowanie go upraszcza proces zarządzania kontenerami, a także zwiększa dostępność i możliwość wykonywania operacji pomiędzy kontenerami.

Rozdział 8. „Portainer” przedstawia graficzny interfejs sieciowy środowiska Docker ułatwiający zarządzanie i sterowanie konfiguracją. Umożliwia on uruchamianie i kasowanie kontenerów, zarządzanie hostami, na których uruchamiasz kontenery, a także pozwala na ocenę ogólnej wydajności pracy środowiska.

Rozdział 9. „Rancher” opisuje zagadnienia związane z narzędziem przeznaczonym do uruchamiania różnych klastrów hostów, a także zarządzania hostami i kontenerami Dockera.

Rozdział 10. „Usługa Docker Cloud” przedstawia produkt, który umożliwia zarządzanie środowiskiem Dockera za pośrednictwem przeglądarki internetowej. Hosting ten wyróżnia fakt, że jest on zapewniany przez twórców Dockera i nie wymaga przeprowadzania skomplikowanej konfiguracji.

Rozdział 11. „Bezpieczeństwo platformy Docker” skupia się na zagadnieniach związanych z bezpieczeństwem Dockera. Tak naprawdę to jeden z najważniejszych problemów każdego serwera i kontenera. Opiszymy dobre praktyki, rzeczy, o których warto pamiętać, i przyjrzymy się różnym problemom dotyczącym bezpieczeństwa. Wspomnimy również o nowym narzędziu, które umożliwia sprawdzanie bezpieczeństwa konfiguracji kontenerów.

Rozdział 12. „Przepływy zadań w platformie Docker” łączy całą zdobytą wcześniej wiedzę i pokazuje, jak zastosować ją w praktyce, wdrażając kontenery w środowisku produkcyjnym. Ponadto w czasie jego lektury dowiesz się, jak monitorować działanie kontenerów, i poznasz zagadnienia związane z zabezpieczaniem kontenerów w celu ułatwienia ich odtwarzania. Dodałkowo przyjrzymy się korzystaniu z zewnętrznych platform Kubernetes i Amazon ECS.

Rozdział 13. „Dalsze kroki z Dockerem” przedstawia zagadnienia dotyczące ulepszania działania aplikacji działającej w kontenerze za pomocą techniki odkrywania usług. Ponadto przyjrzymy się kierunkom rozwoju Dockera. Dowiesz się, jak możesz wnieść własne ulepszenia do tej platformy i dołączyć do rozwijającej ją społeczności.

Czego potrzebujesz?

Będziesz musiał zainstalować i skonfigurować Dockera w wersji 17.03 (CE) w jednym z następujących systemów operacyjnych:

- Windows 10 Professional,
- macOS Sierra,
- Ubuntu 16.04 LTS.

Ponadto powinieneś dysponować dostępem do publicznych platform, takich jak DigitalOcean lub Amazon Web Services.

Do kogo adresowana jest ta książka?

Książka jest adresowana do programistów, administratorów, a także innych osób chcących rozwijać dalej swoje umiejętności pracy w Dockerze. Powinieneś dysponować podstawową wiedzą na temat Dockera, ale pomimo tego na początku książki postanowiliśmy Ci je przyпомнić w celu ułatwienia przyswajania nowych wiadomości.

Konwencje typograficzne przyjęte w tej książce

W książce zastosowano różne czcionki w celu odróżnienia od siebie różnych treści.

Elementy kodu wplecone w treść akapitów wyróżniono czcionką o stałej szerokości znaków.

Dłuższe bloki kodu wyróżniono charakterystycznym wcięciem i czcionką o stałej szerokości znaków:

```
events {  
    worker_connections 1024;  
}
```

Elementy bloku kodu, na które chcielibyśmy zwrócić szczególną uwagę, wyróżniono pogrubieniem:

```
events {  
    worker_connections 1024;  
}
```

Polecenia, które należy wprowadzić w wierszu poleceń, wyróżniono znakiem \$ i czcionką o stałej szerokości znaków:

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

Nowe terminy i ważne słowa kluczowe wyróżniono **czcionką pogrubioną**. Nazwy elementów wyświetlanych na ekranie, takich jak przyciski i elementy interfejsu, a także ścieżki dostępu, wyróżniono *kursywą*.

Wskaźówki, ostrzeżenia lub ważne uwagi pojawiają się w takich ramkach.

Dodatkowe materiały do pobrania

Na stronie <ftp://ftp.helion.pl/przyklady/dockaz.zip> znajdziesz archiwum zawierające pliki z kodem źródłowym zaprezentowanym w tej książce oraz kolorowe wersje wszystkich rysunków.

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 rozwiązywanie 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 pobrania 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. Zaczniemy 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:

NA DEVIE DZIAŁAŁO



SKĄD WZIAŁ SIĘ TEN PROBLEM?

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 to 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.

Rozwiążanie 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 opiszymy w rozdziale 2. „Tworzenie obrazów kontenerów”) lub pliku Docker Compose (zagadnienie to opiszymy w rozdziale 6. „Docker Compose”).

Dzięki temu rozwiążaniu 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 wykonywalem 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 nadziedzny 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.

Rozwiązywanie problemu za pomocą Dockera

Zastosowanie Dockera to doskonale rozwiązywanie 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 wiążą 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 zawieszane 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 udostępnii 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.

Rozwiążanie 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.

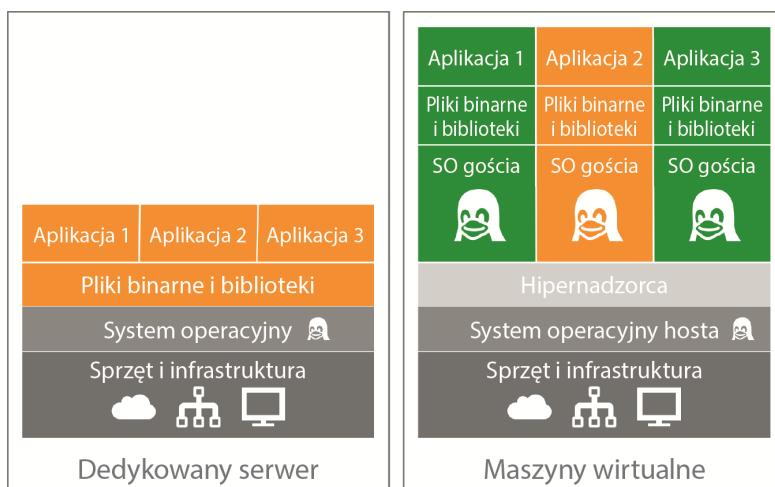
Różnice pomiędzy dedykowanymi hostami, maszynami wirtualnymi i Dockerem

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

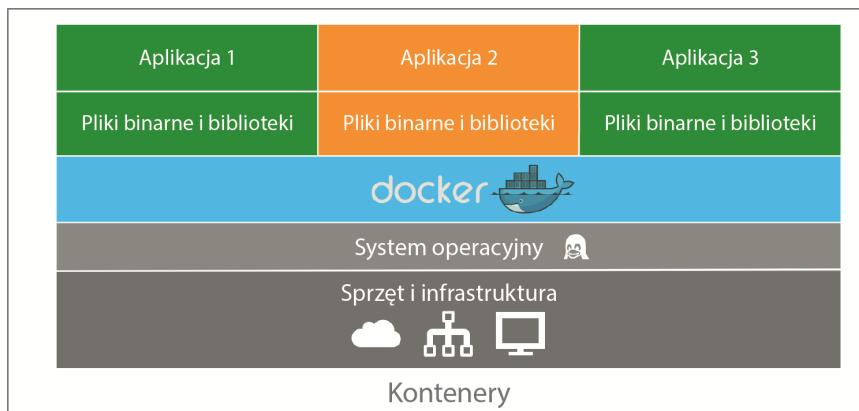
Docker to system zarządzania, który ułatwia obsługę kontenerów **LXC (Linux Containers)**. Rozwiązanie to umożliwia tworzenie obrazów w środowiskach wirtualnych na laptopach programistów i uruchamianie na nich poleceń. Docker umożliwia wykonywanie na kontenerach uruchomionych lokalnie tych samych operacji,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

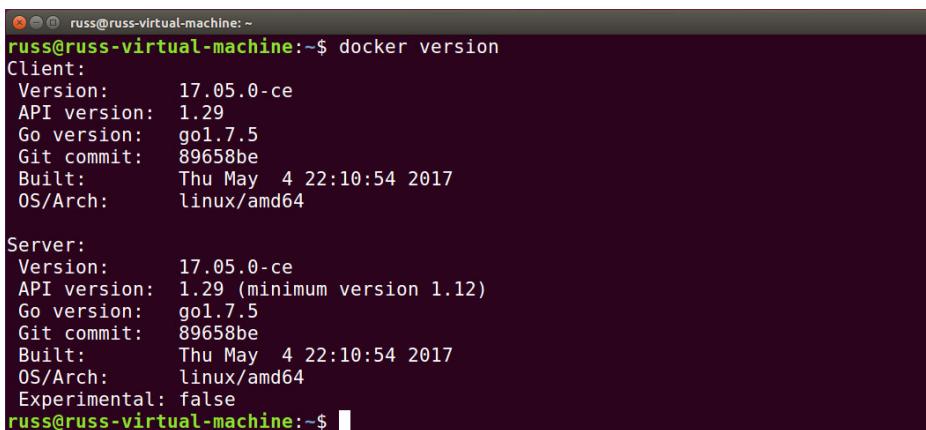
```
$ curl -sSL https://get.docker.com/ | sh  
$ sudo systemctl start docker
```

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:



```
russ@russ-virtual-machine:~  
russ@russ-virtual-machine:~$ docker version  
Client:  
  Version:      17.05.0-ce  
  API version:  1.29  
  Go version:   go1.7.5  
  Git commit:   89658be  
  Built:        Thu May 4 22:10:54 2017  
  OS/Arch:      linux/amd64  
  
Server:  
  Version:      17.05.0-ce  
  API version:  1.29 (minimum version 1.12)  
  Go version:   go1.7.5  
  Git commit:   89658be  
  Built:        Thu May 4 22:10:54 2017  
  OS/Arch:      linux/amd64  
  Experimental: false  
russ@russ-virtual-machine:~$
```

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 znajdującej 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:

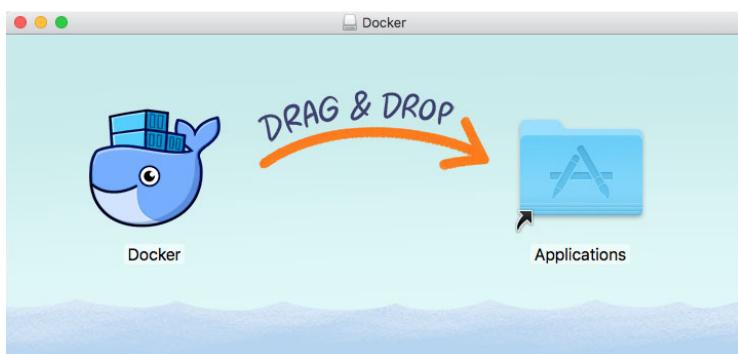
```
$ docker-compose version
$ docker-machine version
```

Instalacja w systemie macOS

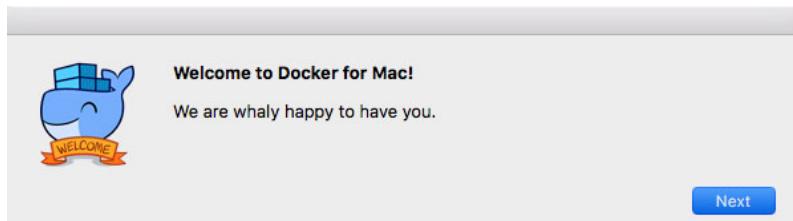
Docker w systemie macOS instalowany jest za pomocą graficznego instalatora.

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:

A screenshot of a terminal window titled "1. russ (bash)". The window displays the output of the "docker version" command. It shows two sections: "Client" and "Server". The Client section includes details like Version: 17.03.1-ce, API version: 1.27, Go version: go1.7.5, Git commit: c6d412e, Built: Tue Mar 28 00:40:02 2017, and OS/Arch: darwin/amd64. The Server section includes details like Version: 17.03.1-ce, API version: 1.27 (minimum version 1.12), Go version: go1.7.5, Git commit: c6d412e, Built: Fri Mar 24 00:00:50 2017, OS/Arch: linux/amd64, and Experimental: true. The prompt "russ in ~" is visible at the bottom left, and a small red "⚡" icon is at the bottom right.

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 — hipernadzorcę 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.

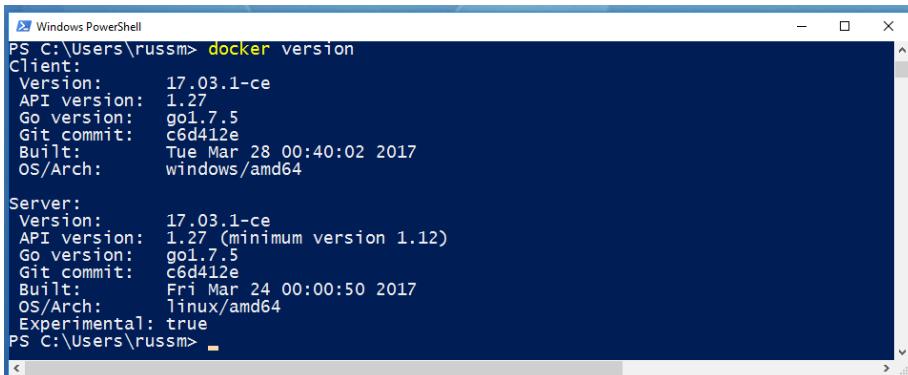
Po zakończeniu procesu instalacji ikona Dockera pojawi się w zasobniku systemowym wyświetlonym w prawym dolnym rogu ekranu. Kliknij ją i wybierz opcję *About Docker*, co spowoduje wyświetlenie okna z informacjami o zainstalowanej wersji Dockera:



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:



```
PS C:\Users\russm> docker version
Client:
  Version: 17.03.1-ce
  API version: 1.27
  Go version: go1.7.5
  Git commit: c6d412e
  Built: Tue Mar 28 00:40:02 2017
  OS/Arch: windows/amd64

Server:
  Version: 17.03.1-ce
  API version: 1.27 (minimum version 1.12)
  Go version: go1.7.5
  Git commit: c6d412e
  Built: Fri Mar 24 00:00:50 2017
  OS/Arch: linux/amd64
  Experimental: true
PS C:\Users\russm>
```

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 zauważyleś, że po uruchomieniu polecenia:

```
$ docker version
```

w przypadku zaprezentowanych trzech instalacji wyświetlane 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 montażu 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 polecień, które są używane podczas zarządzania obrazami, a na koniec zajmiemy się poleceniami dotyczącymi kontenerów.

Docker niedawno zmienił strukturę klienta obsługiwanej 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 obsługiwany 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 polecień Dockera wraz z krótkimi opisami ich funkcji. W celu uzyskania bardziej szczegółowych informacji dotyczących wybranego polecenia wpisz:

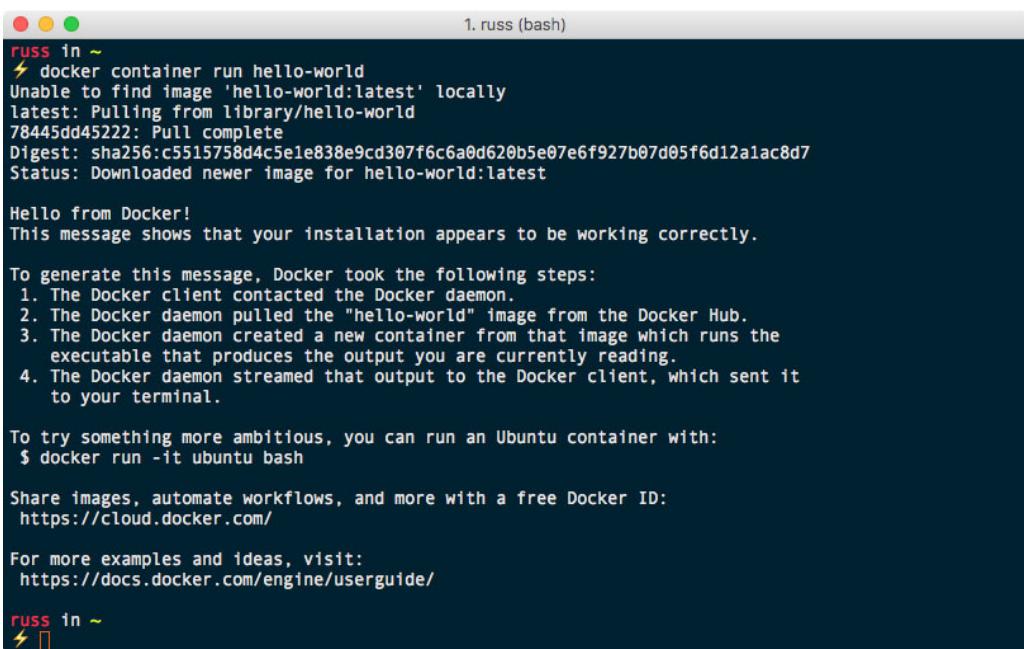
```
$ docker <POLECENIE> --help
```

Teraz możemy przystąpić do uruchomienia kontenera `hello-world`. Zróbmy to za pomocą polecenia (w zależności od konfiguracji systemu Linux może istnieć konieczność poprzedzenia polecenia 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 znajdującej się w nim kodu.

W oknie Terminala powinny zostać wyświetlane następujące komunikaty:



```
russ in ~
$ docker container run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Pull complete
Digest: sha256:c5515758d4c5ele838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
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.
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://cloud.docker.com/

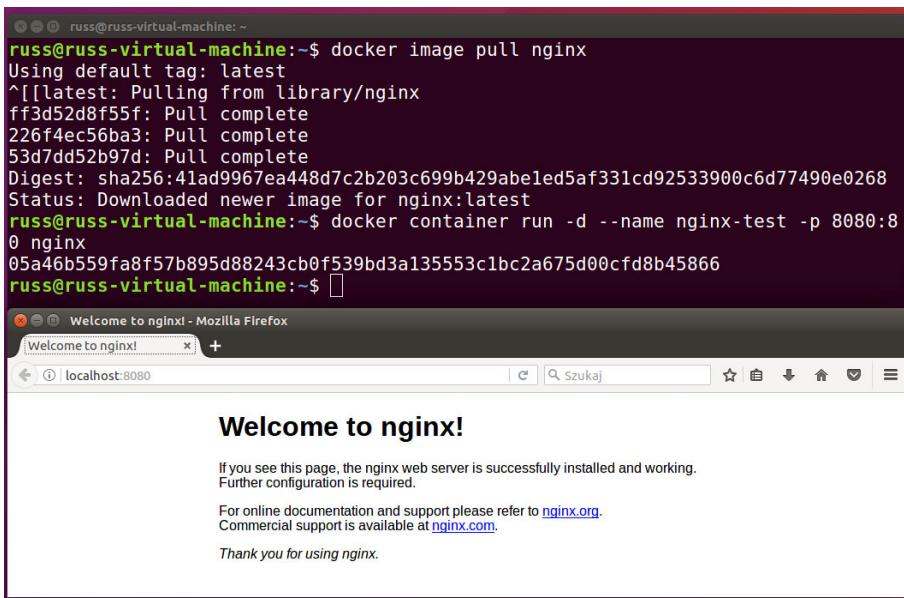
For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/
```

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 `nginx-test` na bazie pobranego obrazu `nginx`. 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:



```
russ@russ-virtual-machine:~$ docker image pull nginx
Using default tag: latest
^[[[latest: Pulling from library/nginx
ff3d52d8f55f: Pull complete
226f4ec56ba3: Pull complete
53d7dd52b97d: Pull complete
Digest: sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268
Status: Downloaded newer image for nginx:latest
russ@russ-virtual-machine:~$ docker container run -d --name nginx-test -p 8080:80
nginx
05a46b559fa8f57b895d88243cb0f539bd3a135553c1bc2a675d00cf8b45866
russ@russ-virtual-machine:~$ 
```

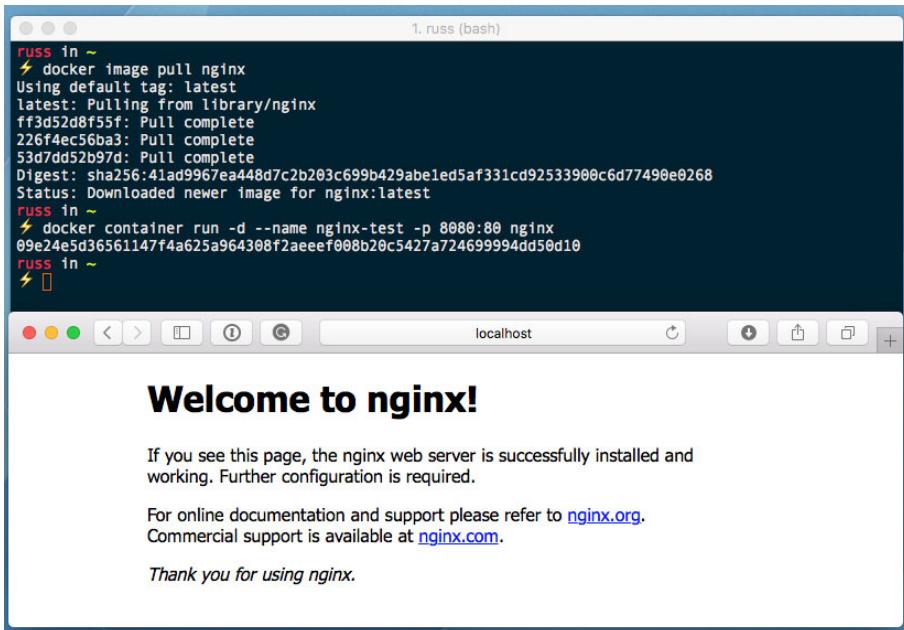
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working.
Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Oto zrzut z systemu macOS:



```
russ in ~
$ docker image pull nginx
Using default tag: latest
latest: Pulling from library/nginx
ff3d52d8f55f: Pull complete
226f4ec56ba3: Pull complete
53d7dd52b97d: Pull complete
Digest: sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268
Status: Downloaded newer image for nginx:latest
russ in ~
$ docker container run -d --name nginx-test -p 8080:80 nginx
09e24e5d36561147f4a625a964308f2aeeef008b20c5427a724699994dd50d10
russ in ~
$ 
```

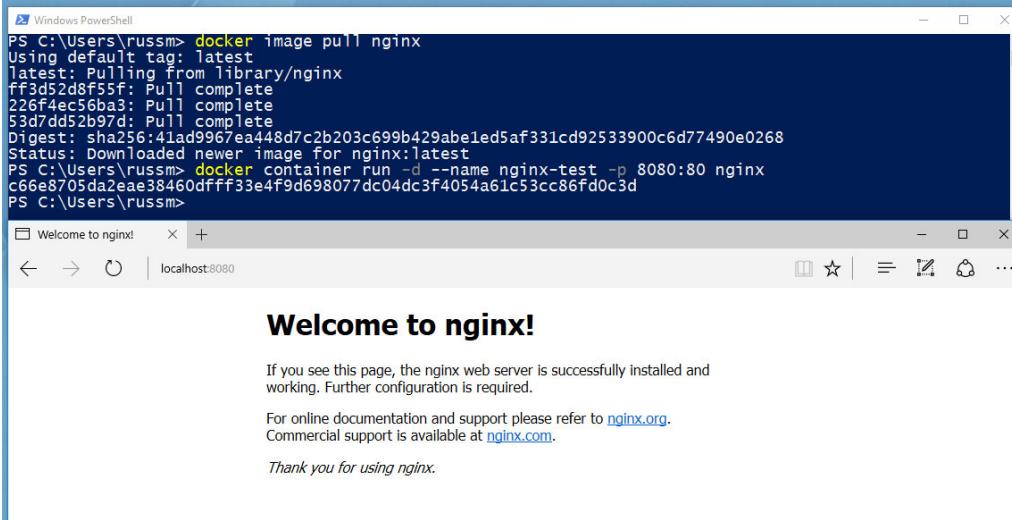
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

A to zrzut z systemu Windows:



```
PS C:\Users\russm> docker image pull nginx
Using default tag: latest
latest: Pulling from library/nginx
ff3d52d8f55f: Pull complete
226f4ec56ba3: Pull complete
53d7dd52b97d: Pull complete
Digest: sha256:41ad9967ea448d7c2b203c699b429abe1ed5af331cd92533900c6d77490e0268
Status: Downloaded newer image for nginx:latest
PS C:\Users\russm> docker container run -d -name nginx-test -p 8080:80 nginx
c66e8705da2eae38460dff33e4f9d698077dc04dc3f4054a61c53cc86fd0c3d
PS C:\Users\russm>
```

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

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

```
$ 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 opiszymy 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 zauważłeś, 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, opisujemy zagadnienia związane z plikami Dockerfile, przechowywaniem obrazów, a także korzystaniem ze zmiennych środowiskowych i wolumenów Dockera.

Tworzenie obrazów kontenerów

Cieszę się, że zdecydowałeś się rozpocząć lekturę tego rozdziału. Za chwilę dowiesz się, jak tworzyć i uruchamiać własne obrazy, a także jak korzystać ze spersonalizowanych zmiennych środowiskowych i skryptów. Oto lista tematów, które poruszmy w tym rozdziale:

- plik Dockerfile;
- proces budowy obrazów;
- tworzenie obrazów na podstawie plików Dockerfile;
- zmienne środowiskowe;
- łączenie powyższych rozwiązań.

Plik Dockerfile — wprowadzenie

W tym podrozdziale opiszemy dokładnie zagadnienia związane z plikiem Dockerfile, a także najlepsze praktyki korzystania z niego. Czym jest plik Dockerfile? Jest to zasadniczo zwykły plik tekstowy zawierający polecenia zdefiniowane przez użytkownika, a wykonywane przez polecenie `docker image build`, które służy do generowania obrazów kontenerów (polecenie to zostanie opisane w dalszej części tego rozdziału).

Oto przykładowa zawartość pliku Dockerfile:

```
FROM alpine:latest
LABEL maintainer="Russ Kendrick <russ@mckendrick.io>"
LABEL description="Plik demonstracyjny - instalator obrazu NGINX."
```

```
RUN apk add --update nginx && \
    rm -rf /var/cache/apk/* && \
    mkdir -p /tmp/nginx/

COPY files/nginx.conf /etc/nginx/nginx.conf
COPY files/default.conf /etc/nginx/conf.d/default.conf
ADD files/html.tar.gz /usr/share/nginx/

EXPOSE 80/tcp

ENTRYPOINT ["nginx"]
CMD ["-g", "daemon off;"]
```

Wszystkie pliki opisane w tym rozdziale możesz pobrać ze strony <ftp://ftp.helion.pl/przyklady/dockaz.zip>.

Jak widzisz, funkcje poszczególnych instrukcji polecenia build znajdujących się w tym pliku można określić z łatwością bez dodatkowych wyjaśnień.

Zanim przejdziemy do opisu przedstawionego pliku Dockerfile, chcielibyśmy poruszyć tematykę dystrybucji Alpine Linux.

Alpine Linux to mała, niezależna i niekomercyjna dystrybucja systemu Linux zaprojektowana z myślą o bezpieczeństwie, wydajności i łatwości obsługi. Więcej informacji na temat tego projektu znajdziesz na jego stronie internetowej (<https://www.alpinelinux.org>).

Alpine Linux z powodu swojego rozmiaru i możliwości stał się domyślnym bazowym obrazem oficjalnych obrazów kontenerów udostępnianych przez autorów Dockera. W związku z tym zdecydowaliśmy się na korzystanie z tej dystrybucji w tej książce. Oto porównanie rozmiaru obrazu Alpine Linux z rozmiarami innych dostępnych dystrybucji:

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|------------|--------|---------------|-------------|---------|
| debian | latest | a25c1eed1c6f | 2 days ago | 123 MB |
| centos | latest | 3bee3060bfcc8 | 4 days ago | 193 MB |
| ubuntu | latest | 7b9b13f7b9c0 | 7 days ago | 118 MB |
| alpine | latest | a41a7446062d | 2 weeks ago | 3.97 MB |
| fedora | latest | 15895ef0b3b2 | 7 weeks ago | 231 MB |

Jak widzisz, Alpine Linux zajmuje zaledwie 3,97 MB. Najwięcej zajmuje obraz dystrybucji Fedora (231 MB). Instalacja systemu Alpine Linux wraz ze wszystkimi pakietami niezbędnymi do samodzielnej pracy serwera zajmuje około 130 MB, czyli wciąż połowę tego, ile wynosi połowa obrazu kontenera dystrybucji Fedora.

Szczegółowa analiza pliku Dockerfile

Przedstawiony wcześniej plik Dockerfile zawierał polecenia:

- FROM,
- LABEL,
- RUN,
- COPY i ADD,
- EXPOSE,
- ENTRYPOINT i CMD,
- inne polecenia pliku Dockerfile.

FROM

Polecenie FROM służy do określania obrazu bazowego. Zgodnie z tym, co pisaliśmy, korzystamy z obrazu Alpine Linux, a więc podajmy nazwę tego obrazu wraz ze znacznikiem interesującej nas wersji. W tym przypadku korzystamy z najnowszego oficjalnego obrazu Alpine Linux, a więc poleceniu FROM przypisaliśmy argumenty `alpine:latest`.

LABEL

Polecenie LABEL może zostać użyte w celu dodania do obrazu dodatkowych informacji. Mogą one mieć dowolną treść, opisywać obraz lub zawierać numer jego wersji. Zaleca się ograniczenie liczby używanych etykiet. Dobra struktura etykiet pomoże innym osobom korzystającym później z wygenerowanego obrazu. Użycie zbyt dużej liczby etykiet może pogorszyć wydajność obrazu, a więc polecam stosowanie się do zasad nadawania etykiet opisanych na stronie <http://label-schema.org>. Polecenie docker inspect umożliwia podgląd etykiet kontenera:

```
$ docker image inspect <IDENTYFIKATOR_KONTENERA>
```

W celu wyświetlenia samych etykiet możesz skorzystać z następującego polecenia:

```
$ docker image inspect -f {{.Config.Labels}} <IDENTYFIKATOR_KONTENERA>
```

W zaprezentowanym przykładzie utworzyliśmy dwie etykiety. Pierwsza z nich określa osobę odpowiedzialną za utrzymanie obrazu: `maintainer="Russ Kendrick <russ@mckendrick.io>"`. Druga etykieta zawiera opis obrazu: `description="Plik demonstracyjny - instalator obrazu NGINX."`.

Ogólnie rzecz biorąc, lepiej jest definiować etykiety podczas tworzenia kontenera na bazie obrazu, a nie podczas budowy obrazu. W związku z tym najlepiej jest zminimalizować liczbę etykiet i umieścić w nich tylko podstawowe informacje opisujące obraz.

RUN

Polecenie RUN służy do nawiązywania interakcji z zawartością obrazu — umożliwia instalację oprogramowania, a także uruchamianie skryptów i poleceń. Zaprezentowane w przykładzie polecenie RUN uruchamia trzy instrukcje:

```
RUN apk add --update nginx && \
    rm -rf /var/cache/apk/* && \
    mkdir -p /tmp/nginx/
```

Pierwsze z tych trzech poleceń składowych (`apk add --update nginx`) instaluje aplikację NGINX za pomocą menedżera pakietów systemu Alpine Linux. Operator `&&` służy do przejścia do kolejnego polecenia, gdy pierwsze zostanie wykonane poprawnie. W celu zwiększenia czytelności pliku zastosowaliśmy operator `\` umożliwiający podział polecenia RUN na kilka linii zawierających poszczególne polecenia składowe. Kolejne polecenie (`rm -rf /var/cache/apk/*`) usuwa wszelkie pliki tymczasowe w celu zminimalizowania rozmiaru obrazu. Ostatnie polecenie składowe (`mkdir -p /tmp/nginx/`) tworzy folder o nazwie `/tmp/nginx/`, co pozwoli na prawne uruchomienie aplikacji NGINX.

Tę część pliku Dockerfile moglibyśmy zapisać również w następujący sposób:

```
RUN apk add --update nginx
RUN rm -rf /var/cache/apk/*
RUN mkdir -p /tmp/nginx/
```

Jednak podobnie jak jest w przypadku wielu etykiet, takie rozwiązywanie spowodowałoby utworzenie warstw dla poszczególnych polecień RUN, czego lepiej jest unikać. Niemniej taki sposób zapisu jest czasem pożądany. Więcej informacji na ten temat znajdziesz w dalszej części tego rozdziału.

COPY i ADD

Na pierwszy rzut oka może się wydawać, że polecenia COPY i ADD działają tak samo, ale tak naprawdę są pomiędzy nimi ważne różnice. Polecenie COPY jest łatwiejsze w użyciu:

```
COPY files/nginx.conf /etc/nginx/nginx.conf
COPY files/default.conf /etc/nginx/conf.d/default.conf
```

Jak się zapewne domyślasz, polecenia te kopią dwa pliki z folderu `files` znajdującego się w systemie plików hosta, w którym przeprowadzany jest proces budowy obrazu. Pierwszy z tych plików to `nginx.conf`. Zawiera on podstawową konfigurację aplikacji NGINX:

```
user nginx;
worker_processes 1;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}
```

```

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    log_format main '$remote_addr - $remote_user
[$time_local] "$request"
$status $body_bytes_sent
$http_referer'
"$http_user_agent"
$http_x_forwarded_for';
    access_log /var/log/nginx/access.log main;
    sendfile off;
    keepalive_timeout 65;
    include /etc/nginx/conf.d/*.conf;
}

```

Operacja kopiowania tego pliku nadpisze domyślną konfigurację aplikacji NGINX, która została przyjęta po zainstalowaniu jej za pomocą polecenia RUN. Kolejny kopiowany plik to *default.conf*, definiujący najprostszego wirtualnego hosta, którego możemy skonfigurować. Plik ten zawiera następujący kod:

```

server {
    location / {
        root /usr/share/nginx/html;
    }
}

```

W wyniku operacji kopiowania nadpisywane są utworzone wcześniej pliki, a więc po co używa się polecenia ADD?

```
ADD files/html.tar.gz /usr/share/nginx/
```

Polecenie to dodaje plik o nazwie *html.tar.gz*. W pliku Dockerfile nie ma jednak żadnych poleceń rozpakowujących to archiwum. Jest tak, ponieważ polecenie ADD automatycznie ładuje, rozpakowuje i umieszcza wyjściowe pliki i foldery w określonej lokalizacji. W tym przypadku lokalizacją tą jest folder */usr/share/nginx/*. W ten sposób, zgodnie z definicją podaną w bloku wirtualnego hosta, uzyskujemy główny folder strony internetowej (*/usr/share/nginx/html/*).

Polecenie ADD może być również użyte w celu dodania obiektów ze zdalnych lokalizacji:

```
ADD http://www.myremotesource.com/files/html.tar.gz /usr/share/nginx/
```

Zaprezentowane przykładowe polecenie pobiera archiwum *html.tar.gz* z adresu *http://www.myremotesource.com/files/* i umieszcza je w folderze */usr/share/nginx/* archiwum. Archiwa pobierane ze zdalnych lokalizacji są traktowane jak zwyczajne pliki — nie są rozpakowywane. Warto o tym pamiętać.

EXPOSE

Polecenie EXPOSE informuje platformę Docker o tym, że w momencie wykonywania zawartości obrazu odsłaniany jest określony port i protokół. Polecenie to nie mapuje portu do hosta, ale otwiera port, umożliwiając dostęp do internetu usłudze uruchomionej w kontenerze. W naszym przykładowym pliku Dockerfile umieściliśmy instrukcję otwierającą port 80 przy każdym uruchomieniu obrazu:

```
EXPOSE 80/tcp
```

ENTRYPOINT i CMD

Przewagą wynikającą z korzystania z polecenia ENTRYPOINT, a nie z opisanego dalej polecenia CMD jest to, że polecenie ENTRYPOINT można połączyć z poleceniem CMD. Polecanie ENTRYPOINT może być przydatne jako samodzielna instrukcja, ale pamiętaj o tym, że warto z niego korzystać w ten sposób tylko wtedy, gdy kontener ma być wykonywalny. Korzystając z wiersza poleceń, poza samymi instrukcjami podajesz dodatkowe parametry interpretowane przez te instrukcje. Właśnie w takich sytuacjach korzysta się z samego polecenia ENTRYPOINT.

Jeżeli chcesz określić domyślne polecenie wykonywane wewnątrz kontenera, to możesz zastosować rozwiązańe podobne do tego, które pokazano w poniższym przykładzie, ale musisz przy tym pamiętać o konieczności używania polecenia, które utrzymuje aktywność kontenera.

W naszym pliku Dockerfile zastosowano następujące polecenia:

```
ENTRYPOINT ["nginx"]
CMD ["-g", "daemon off;"]
```

Zaprezentowana para instrukcji sprawia, że przy każdym uruchomieniu kontenera na bazie stworzonego przez nas obrazu uruchomiona zostanie aplikacja nginx (zgodnie z instrukcją ENTRYPOINT), a instrukcja CMD spowoduje wywołanie następującego polecenia:

```
$ nginx -g daemon off;
```

Przyjrzyj się następującej instrukcji:

```
$ docker container run -d --name nginx dockerfile-example -v
```

Daje ona taki sam efekt jak instrukcja:

```
$ nginx -v
```

Spowoduje ona wyświetlenie aktualnej wersji aplikacji NGINX i zatrzymanie działania kontenera, ponieważ aplikacja NGINX zostanie wywołana tylko w celu wyświetlenia numeru wersji, a następnie proces tej aplikacji jest wstrzymywany.

Inne polecenia pliku Dockerfile

Nasz plik Dockerfile nie zawierał wszystkich możliwych polecień. Przyjrzyjmy się innym poleceniom, które można umieścić w pliku Dockerfile.

USER

Instrukcja USER pozwala na określenie nazwy użytkownika używanej podczas uruchamiania różnych poleceń. Instrukcja USER może być użyta w kontekście poleceń RUN, CMD lub ENTRYPPOINT pliku Dockerfile. Użytkownik zdefiniowany w instrukcji USER musi istnieć. W przeciwnym wypadku proces budowy obrazu zostanie zakończony niepowodzeniem. Użycie tej instrukcji może również wywołać problemy związane z uprawnieniami, które mogą być związane z samym kontenerem, a także z montowanymi woluminami.

WORKDIR

Polecenie WORKDIR służy do określania katalogu roboczego dla tego samego zestawu instrukcji, co polecenie USER (RUN, CMD i ENTRYPPOINT). Pozwala ono również na korzystanie z instrukcji CMD i ADD.

ONBUILD

Instrukcja ONBUILD pozwala na zdefiniowanie ciągu poleceń uruchamianych, gdy utworzony przez Ciebie obraz zostanie użyty w charakterze obrazu bazowego kontenera. Jeżeli np. chcesz przekazać obraz zespołowi programistów, a każda z osób pracujących w tym zespole chce sprawdzić działanie innego kodu, to możesz skorzystać z instrukcji ONBUILD w celu przygotowania kontenera do wykonania opracowanego przez nich kodu. Programiści będą mogli dodać swój kod do wskazanego przez Ciebie katalogu, a następnie uruchomić nowe polecenie docker build, które doda ich kod do aktywnego obrazu. Instrukcja ONBUILD może być używana w połączeniu z instrukcjami ADD i RUN. Przyjrzyj się następującemu przykładowi:

```
ONBUILD RUN apk update && apk upgrade && rm -rf /var/cache/apk/*
```

Instrukcja ta uruchomii proces aktualizacji i zmiany wersji na nowszą za każdym razem, gdy obraz będzie używany jako baza innego obrazu.

ENV

Polecenie ENV służy do definiowania zmiennych środowiskowych wewnętrz obrazu, gdy jest on budowany i uruchamiany. Zmienne te mogą zostać nadpisane podczas uruchamiania obrazu.

Dobre praktyki pracy z plikami Dockerfile

Przypomnij sobie podstawowe instrukcje, które mogą być umieszczone w plikach Dockerfile. Teraz chcielibyśmy przedstawić wytyczne dobrych praktyk tworzenia plików Dockerfile:

- Wyrób sobie nawyk korzystania z pliku *.dockerignore*. Plik ten zostanie opisany w kolejnej sekcji. Jeżeli korzystałeś wcześniej z plików *.gitignore*, to zapewne domyślasz się, do czego on służy. Można w nim określić elementy, które zostaną pominięte podczas procesu budowy obrazu.
- Staraj się nie umieszczać w jednym folderze dwóch plików Dockerfile — pomaga to w utrzymaniu porządku.

- Korzystaj z programu do kontroli wersji pliku Dockerfile tak, jakby był to zwyczajny dokument tekstowy. Pozwoli to na ewentualne odtworzenie jego poprzedniej wersji.
- Staraj się minimalizować liczbę pakietów przypadających na jeden obraz. Budowane obrazy powinny być jak najmniejsze. W uzyskaniu jak najmniejszego obrazu z pewnością pomoże nieinstalowanie zbędnych pakietów.
- Uruchamiaj tylko jedną aplikację na kontener. Jeżeli musisz uruchomić kolejną aplikację, to najlepiej, abyś zrobił to w kolejnym kontenerze. W jednym kontenerze możesz uruchomić kilka poleceń, ale najlepiej jest je rozdzielać. Staraj się stosować jak najprostsze rozwiązania. Zbytnia komplikacja pliku Dockerfile może okazać się później problematyczna.
- Ucz się na przykładach. Autorzy Dockera udostępniają poradnik publikowania oficjalnych obrazów umieszczanych w repozytorium Docker Hub. Warto się z nim zapoznać. Dokument ten znajdziesz na stronie <https://github.com/docker-library/official-images/>.

Budowanie obrazów Dockera

W tym podrozdziale zajmiemy się poleceniem `docker build`. Czas zrobić coś praktycznego i zbudować bazę kolejnych obrazów. Dowiesz się, jak wykonać tę operację na wiele sposobów. Obraz wygenerowany w tym podrozdziale możesz traktować jako szablon utworzony za pomocą maszyn wirtualnych. Pomoże on zaoszczędzić czas i część pracy — będziesz mógł się skupić na tworzeniu aplikacji, która będzie mogła być dodana do nowego obrazu.

Polecenie `docker build`

Wiesz już, jak stworzyć poprawny plik Dockerfile, a więc teraz możesz nauczyć się generować na jego podstawie obraz. Polecenie `docker build` obsługuje wiele parametrów. W związku z tym skorzystajmy z argumentu `--help` w celu wyświetlania możliwych opcji:

```
$ docker image build --help
Usage: docker image build [OPTIONS] PATH | URL | -
      Build an image from a Dockerfile
```

Pod tym komunikatem zobaczysz długą listę flag, z których możesz korzystać podczas budowy obrazu.

Budowa obrazu na podstawie pliku Dockerfile

Liczba dostępnych opcji może wydawać się przytłaczająca, ale musimy korzystać tylko z jednej: `--tag` (w skrócie `-t`) służącej do nadawania obrazowi nazwy.

Inne opcje mogą być użyte w celu ograniczenia zasobów procesora i pamięci używanych przez proces budowania. Czasami może istnieć potrzeba zastosowania takiego rozwiązania — proces budowy obrazu może wtedy przebiegać wolniej, ale w przypadku długich procesów przeprowadzanych na komputerze lokalnym lub serwerze produkcyjnym warto korzystać z takich ograniczeń. Zwykle polecenie docker build jest uruchamiane w tym samym folderze, w którym znajduje się plik Dockerfile. W takim przypadku nie używa się flagi definiującej plik `--file (-f)`. Umieszczanie plików Dockerfile w oddzielnych folderach pomaga w ich uporządkowaniu i utrzymaniu konwencji nazewnictwa plików.

Warto pamiętać, że w przypadku przekazania dodatkowych zmiennej środowiskowej w postaci argumentów procesu budowania obrazów zmienne te są używane w procesie budowy. Nie są one dziedziczone przez kontener. Zmienne środowiskowe mogą zawierać np. konfigurację serwera proxy, czyli informacje przydatne tylko w początkowym środowisku, w którym przeprowadzany jest proces budowania lub testowania.

Plik `.dockerignore`

Plik `.dockerignore` jest używany do wybierania plików i folderów, które nie mają wchodzić w skład budowanego obrazu. Domyślnie polecenie docker build umieszcza w obrazie wszystkie pliki umieszczone w folderze pliku Dockerfile. Plik `.dockerignore` powinien znajdować się w tym samym folderze, w którym znajduje się używany przez nas plik Dockerfile (między innymi właśnie dlatego poszczególne pliki Dockerfile powinny być umieszczone w oddzielnych folderach). Umieszczenie w tym samym folderze wszystkich elementów, z których chcesz korzystać w obrazie, pomoże zminimalizować liczbę obiektów wymienionych w pliku `.dockerignore`.

Budowanie własnych obrazów na podstawie plików Dockerfile

Pierwszy sposób tworzenia bazowych obrazów Docker, który chcemy opisać, polega na korzystaniu z plików Dockerfile. Skorzystamy z pliku Dockerfile przedstawionego w poprzednim podrozdziale w celu zbudowania obrazu NGINX za pomocą polecenia docker build. Przyjrzyjmy się jeszcze raz zawartości tego pliku:

```
FROM alpine:latest
LABEL maintainer="Russ Kendrick <russ@mckendrick.io>"
LABEL description="Plik demonstracyjny - instalator obrazu NGINX."

RUN apk add --update nginx && \
    rm -rf /var/cache/apk/* && \
    mkdir -p /tmp/nginx/

COPY files/nginx.conf /etc/nginx/nginx.conf
COPY files/default.conf /etc/nginx/conf.d/default.conf
ADD files/html.tar.gz /usr/share/nginx/

EXPOSE 80/tcp

ENTRYPOINT ["nginx"]
CMD ["-g", "daemon off;"]
```

Budowę obrazu możemy rozpoczęć na dwa sposoby. Pierwszy z nich polega na wybraniu pliku za pomocą argumentu `-f` polecenia `docker build`. Dodatkowo użyjemy flagi `-t` w celu nadania nowemu obrazowi unikalnej nazwy:

```
$ docker image build --file <ściezka_pliku_Dockerfile> --tag  
<REPOZYTORIUM>:<ZNACZNIK> .
```

Argument `<REPOZYTORIUM>` jest zwykle nazwą użytkownika wybraną podczas logowania się do repozytorium Docker Hub (więcej informacji na ten temat znajdziesz w rozdziale 3. „Przechowywanie obrazów i ich dystrybucja”). Na razie będziemy korzystać z lokalnego repozytorium (`local`) i znacznika określającego wersję kontenera lub wyrażenie opisujące jego zawartość:

```
$ docker image build --file /ściezka/twojego/pliku/dockerfile --tag  
local:dockerfile-example .
```

W praktyce zwykle nie korzysta się z argumentu `--file`, co może być problematyczne, jeżeli w tym samym folderze znajdują się inne pliki — pliki, które nie powinny zostać dołączone do budowanego obrazu. Najlepiej jest umieścić plik Dockerfile w oddzielnym folderze, w którym znajdująć się będą tylko pliki, które chcesz umieścić wewnątrz obrazu za pomocą instrukcji ADD lub COPY:

```
$ docker image build --tag local:dockerfile-test .
```

Nie możesz zapomnieć o kropce umieszczonej na końcu. Dzięki niej polecenie `docker build` zbuduje obraz w bieżącym folderze.

Podczas budowania obrazu w oknie Terminala zobaczysz komunikaty podobne do tych z pierwszego rysunku na następnej stronie.

Po zbudowaniu obrazu uruchom polecenie sprawdzające jego dostępność i rozmiar:

```
$ docker image ls
```

Jak widzisz, zbudowany przez mnie obraz zajmuje 5,49 MB (patrz drugi zrzut na następnej stronie).

Korzystanie z utworzonego wcześniej kontenera

Obraz bazowy najprościej jest zdobyć, pobierając go z serwisu Docker Hub. W repozytorium GitHub można znaleźć pliki Dockerfile użyte do wygenerowania tych obrazów. Jak widzisz, są przynajmniej dwa powody, dla których warto korzystać z obrazów przygotowanych przez innych użytkowników Dockera. Dzięki plikom Dockerfile możesz dowiedzieć się, co dokładnie zostało dołączone do obrazu, i ewentualnie dodać do niego to, czego potrzebujesz. Plik Dockerfile możesz włączyć do systemu kontroli wersji i w razie ewentualnej potrzeby wracać do jego starszych wersji.

```
km@km-Extensa-5220: ~/kod/r02/dockerfile-test
km@km-Extensa-5220:~/kod/r02/dockerfile-test$ sudo docker image build --tag local:dockerfile-test .
Sending build context to Docker daemon 64.51kB
Step 1/10 : FROM alpine:latest
--> 3fd9065eaf02
Step 2/10 : LABEL maintainer="Russ McKendrick <russ@mckendrick.io>"
--> Using cache
--> 07923e5a70fe
Step 3/10 : LABEL description="Plik demonstracyjny - instalator obrazu NGINX."
--> Running in c7a64e65b99d
Removing intermediate container c7a64e65b99d
--> 57105a4c98b4
Step 4/10 : RUN apk add --update nginx && rm -rf /var/cache/apk/* &&
mkdir -p /tmp/nginx/
--> Running in 410d43a0ac9a
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/2) Installing pcre (8.41-r1)
(2/2) Installing nginx (1.12.2-r3)
Executing nginx-1.12.2-r3.pre-install
Executing busybox-1.27.2-r7.trigger
OK: 6 MiB in 13 packages
Removing intermediate container 410d43a0ac9a
--> 770c04e3a233
Step 5/10 : COPY files/nginx.conf /etc/nginx/nginx.conf
--> 9842ceaacc5e
Step 6/10 : COPY files/default.conf /etc/nginx/conf.d/default.conf
--> 58ca7b205988
Step 7/10 : ADD files/html.tar.gz /usr/share/nginx/
--> 6ef70f8f9f91
Step 8/10 : EXPOSE 80/tcp
--> Running in 0508e57ac767
Removing intermediate container 0508e57ac767
--> b1d14edfc914
Step 9/10 : ENTRYPOINT ["nginx"]
--> Running in f1a577bfdf89
Removing intermediate container f1a577bfdf89
--> d01c3ba816b0
Step 10/10 : CMD ["-g", "daemon off;"]
--> Running in dc0836ab68e9
Removing intermediate container dc0836ab68e9
--> 7673e6687271
Successfully built 7673e6687271
Successfully tagged local:dockerfile-test
km@km-Extensa-5220:~/kod/r02/dockerfile-test$
```

```
russ in ~/Desktop/dockerfile-example
⚡ docker image ls
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
local           dockerfile-example   c64e6e3c31d5    2 minutes ago  5.49 MB
alpine          latest        a41a7446062d    2 weeks ago   3.97 MB
russ in ~/Desktop/dockerfile-example
⚡
```

Istnieje jeszcze jedno rozwiązanie, którego jednak nie polecam, bo nie uważam go za dobrą praktykę, i sądzę, że nie powinieneś z niego korzystać. Moim zdaniem nadaje się ono tylko do sprawdzania poprawności działania polecień w trakcie prototypowania, przed umieszczeniem ich w pliku Dockerfile. Rozwiązanie to polega na interaktywnym uruchamianiu polecień w powłoce.

Demonstrację tej techniki zacznijmy od pobrania obrazu, który chcemy zastosować w roli bazy. Ponownie skorzystajmy z systemu Alpine Linux:

```
$ docker image pull alpine:latest
```

Teraz możemy uruchomić kontener w celu dodania do niego pakietów:

```
$ docker container run -it --name alpine-test alpine /bin/sh
```

Po uruchomieniu kontenera możesz dodać do niego pakiety. W tym wypadku skorzystamy z polecenia apk, ale możesz skorzystać również z innych, preferowanych przez siebie poleceń służących do zarządzania pakietami w systemie Linux.

Oto przykładowe polecenia służące do zainstalowania aplikacji NGINX:

```
$ apk update  
$ apk upgrade  
$ apk add --update nginx  
$ rm -rf /var/cache/apk/*  
$ mkdir -p /tmp/nginx/  
$ exit
```

Po zainstalowaniu niezbędnych pakietów musimy zapisać kontener. Polecenie exit znajdujące się na końcu zaprezentowanego ciągu poleceń zatrzyma działanie kontenera, ponieważ proces powłoki, z którego wychodzimy, jest jednocześnie procesem utrzymującym aktywność kontenera.

Zatrzymaj się na chwilę! Nie polecam używania zaprezentowanych przed chwilą polecień w celu tworzenia i udostępniania obrazów. Jest to tylko jedna z teoretycznie możliwych do zastosowania technik, które chciałem przedstawić w tym rozdziale.

W celu zapisania zatrzymanego kontenera jako obrazu musimy skorzystać z polecenia o składni:

```
$ docker container commit <ścieżka_pliku_Dockerfile> <REPOZYTORIUM>:<ZNACZNIK>
```

Oto przykład polecenia zapisującego kopię uruchomionego i zmodyfikowanego kontenera:

```
$ docker container commit alpine-test local:broken-container
```

Kontenerowi nadałem nazwę broken-container (uszkodzony kontener). Zrobiłem to dlatego, że jednym z podstawowych zastosowań tej techniki jest zapisywanie problematycznych kontenerów w formie obrazów lub plików TAR w celu udostępnienia ich innym osobom, które mogą pomóc w znalezieniu źródła problemu.

W celu zapisania pliku obrazu należy skorzystać z polecenia o następującej składni:

```
$ docker image save -o <nazwa_pliku.tar> <REPOZYTORIUM>:<ZNACZNIK>
```

Oto polecenie, które pozwoli na zapisanie naszego kontenera:

```
$ docker image save -o broken-container.tar local:broken-container
```

Polecenie to wygenerowało plik *broken-container.tar* zajmujący 6,4 MB. Osoba dysponująca tym plikiem może go rozpakować i przyjrzeć się jego zawartości, co wynika ze struktury widocznej na poniższym rysunku:

```
russ in ~/Desktop/broken-container
⚡ tree
.
├── 2a640dea2e1165b63947335c22f1f70163730f21a70e5c121681f8da0c75d935.json
└── a90a4db4af2a27c91da87db576184e7876798970b138cb3f91c484d947fd3456
    ├── VERSION
    └── json
        └── layer.tar
├── aa49f0f463a3618c10e90c3bf737207b8b1181432d7bd634d4e297b4af0fcba3
    ├── VERSION
    └── json
        └── layer.tar
└── manifest.json
└── repositories

2 directories, 9 files
russ in ~/Desktop/broken-container
⚡
```

Obraz jest zbiorem plików JSON, folderów i innych archiwów TAR. Wszystkie obrazy mają podobną strukturę, a więc być może zastanawiasz się, dlaczego ta metoda jest zła.

Największym czynnikiem wpływającym na to jest zaufanie — użytkownik obrazu wygenerowanego w ten sposób nie będzie mógł łatwo sprawdzić, jaki obraz uruchamia. Czy byłbyś skłonny pobrać obraz z nieznanego źródła bez sprawdzenia, jak został on zbudowany? W takiej sytuacji nie wiedziałbyś, jak go skonfigurowano i jakie pakiety do niego dodano.

Ponadto podczas budowania obrazu w ten sposób nie możesz skorzystać z poleceń takich jak `ENTRYPOINT`, `CMD` i `EXPOSE`, a więc nie możesz przygotować domyślnej konfiguracji środowiska. Użytkownik takiego obrazu będzie musiał definiować wszystko samodzielnie w poleceniu `docker container run`.

Tego typu obrazy były popularne zaraz po powstaniu platformy Docker. Tak naprawdę sam częściowo za to odpowiadam. W tamtych czasach uruchomienie „maszyny” i utworzenie na jej podstawie „kopii matki” wydawało się dość sensowne, ale przez ostatnich kilka lat Docker rozszerzył funkcje przeznaczone do budowy obrazów do tego stopnia, że rozważanie korzystania z zaprezentowanej techniki nie ma już nawet sensu.

Budowanie od podstaw

Dotychczas w roli bazowych obrazów stosowaliśmy gotowe rozwiązania pobrane z serwisu Docker Hub, ale nie musisz korzystać z tego rozwiązania — obrazy mogą być budowane od podstaw.

Zwykłe określenie „od podstaw” oznacza rozpoczęwanie pracy od początku, bez jakiegokolwiek bazy. Zabieg taki może mieć wiele zalet, ponieważ umożliwia tworzenie bardzo małych obrazów, ale jeżeli nie masz większego doświadczenia w pracy z Dockerem, to może on wydawać się bardzo skomplikowany.

Twórcy Dockera udostępnili w serwisie Docker Hub pusty plik TAR o nazwie `scratch`. Można z niego skorzystać w sekcji `FROM` pliku Dockerfile. Możesz oprzeć na nim cały budowany obraz i dodać do niego niezbędne komponenty.

Ponownie spróbujmy stworzyć obraz oparty na systemie operacyjnym Alpine Linux. Wynika to z tego, że system ten jest udostępniany nie tylko w formie obrazu ISO, obrazu Dockera i obrazów różnych maszyn wirtualnych, ale jest on również udostępniany w formie archiwum TAR, które można pobrać z repozytorium znajdującego się na stronie internetowej tej dystrybucji. Wybierz właściwy plik — pisząc tę książkę, pobraliśmy wersję `x86_64` z sekcji `MINI ROOT FILESYSTEM` znajdującej się pod adresem <https://www.alpinelinux.org/downloads/>.

Po pobraniu archiwum musisz utworzyć plik Dockerfile korzystający z bazy w postaci pliku `scratch`, a następnie dodający do niej pobrane przed chwilą archiwum TAR. Możesz to zrobić za pomocą następującego kodu:

```
FROM scratch
ADD files/alpine-minirootfs-3.6.1-x86_64.tar /
CMD ["/bin/sh"]
```

Dysponując plikiem Dockerfile i systemem operacyjnym w postaci archiwum TAR, możesz zbudować obraz Dockera za pomocą następującego polecenia (obrazowi nadaliśmy nazwę `fromscratch` — czyli „od podstaw”):

```
$ docker image build --tag local:fromscratch .
```

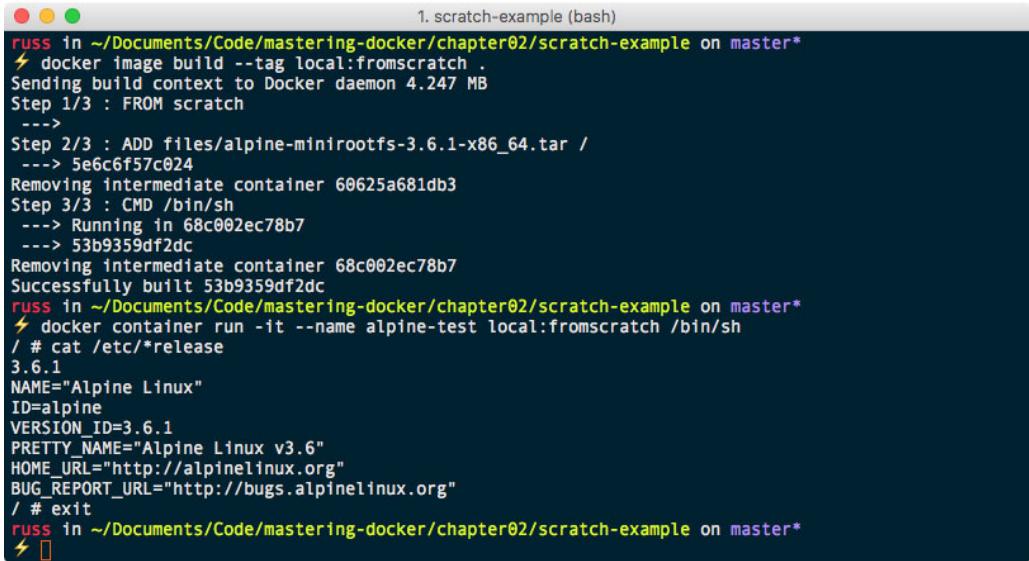
Po zbudowaniu obrazu możesz go przetestować za pomocą następującego polecenia:

```
$ docker container run -it --name alpine-test local:fromscratch /bin/sh
```

Polecenie to powinno uruchomić powłokę systemu Alpine Linux umieszczonego w obrazie. Możesz to sprawdzić, uruchamiając polecenie:

```
$ cat /etc/*release
```

Spowoduje to wyświetlenie informacji na temat wersji uruchomionego kontenera. Oto dane wyświetlane w oknie Terminala, które przedstawiają proces budowy obrazu:



```
1. scratch-example (bash)
russ in ~/Documents/Code/mastering-docker/chapter02/scratch-example on master*
⚡ docker image build --tag local:fromscratch .
Sending build context to Docker daemon 4.247 MB
Step 1/3 : FROM scratch
-->
Step 2/3 : ADD files/alpine-minirootfs-3.6.1-x86_64.tar /
--> 5e6c6f57c024
Removing intermediate container 60625a681db3
Step 3/3 : CMD /bin/sh
--> Running in 68c002ec78b7
--> 53b9359df2dc
Removing intermediate container 68c002ec78b7
Successfully built 53b9359df2dc
russ in ~/Documents/Code/mastering-docker/chapter02/scratch-example on master*
⚡ docker container run -it --name alpine-test local:fromscratch /bin/sh
/ # cat /etc/*release
3.6.1
NAME="Alpine Linux"
ID=alpine
VERSION_ID=3.6.1
PRETTY_NAME="Alpine Linux v3.6"
HOME_URL="http://alpinelinux.org"
BUG_REPORT_URL="http://bugs.alpinelinux.org"
/ # exit
russ in ~/Documents/Code/mastering-docker/chapter02/scratch-example on master*
⚡
```

Wszystko wydaje się proste. Prostota przygotowania obrazu od podstaw wynika ze sposobu dostarczenia systemu Alpine Linux, a więc tworzenie obrazów od podstaw nie zawsze jest takie proste.

Oto wybrane narzędzia, które mogą być użyte w celu przygotowania systemu operacyjnego do umieszczenia go w obrazie Dockera:

- **Debootstrap:** <https://wiki.debian.org/Debootstrap/>;
- **Yumbootstrap:** <https://github.com/dozzie/yumbootstrap/>;
- **Rinse:** <http://collab-maint.alioth.debian.org/rinse/>;
- **Docker contrib scripts:** <https://github.com/moby/moby/tree/master/contrib/>.

Nie będziemy zagłębiać się w szczegóły użytkowania tych skryptów, ponieważ jeżeli chcesz tworzyć obrazy od podstaw, to znaczy, że najprawdopodobniej musisz sprostać jakimś nie-standardowym wymaganiom.

Jakiego typu mogą to być wymagania? W większości przypadków będzie to potrzeba uruchomienia na nowoczesnej platformie aplikacji korzystających z przestarzałych technologii, takich jak np. niewspierane już systemy operacyjne, których nie można pobrać z serwisu Docker Hub. W takim przypadku można uruchomić obraz, zainstalować w nim aplikację, a następnie uruchomić ten kontener w nowszym systemie.

Zmienne środowiskowe

W tym podrozdziale zajmiemy się zmiennymi środowiskowymi. Zmienne te pozwalają na wykonanie wielu operacji z poziomu pliku Dockerfile. Jeżeli umiesz kodować, to z pewnością zechcesz z nich korzystać.

Dla osób, które nie mają doświadczenia w programowaniu, korzystanie ze zmiennych środowiskowych może na początku wydawać się trudne, ale zaraz po opanowaniu stają się one niezwykle przydatne. Zmiennych tych można używać w celu definiowania pewnych parametrów podczas uruchamiania kontenera — minimalizują one konieczność aktualizowania wielu poleceń plików Dockerfile i skryptów uruchamianych na serwerze.

Umieszczanie zmiennych środowiskowych w pliku Dockerfile

Zmienne środowiskowe można definiować w pliku Dockerfile za pomocą instrukcji ENV. Oto jej składnia:

```
ENV <klucz> <wartość>
ENV username admin
```

Klucz i wartość można również rozdzielić znakiem równości:

```
ENV <klucz>=<wartość>
ENV username=admin
```

Dlaczego istnieją dwie notacje? Pierwsza notacja umożliwia definiowanie po jednej zmiennej środowiskowej w każdej linii pliku Dockerfile, a druga notacja umożliwia definiowanie wielu zmiennych środowiskowych w tej samej linii:

```
ENV username=admin database=db1 tableprefix=pr2_
```

Polecenie Docker inspect umożliwia podgląd zmiennych środowiskowych przypisanych do wybranego obrazu:

```
$ docker image inspect <IDENTYFIKATOR_OBRAZU>
```

Wiesz już, jak definiować zmienne środowiskowe. Czas przyjrzeć się ich zastosowaniom.

Czas wszystko połączyć ze sobą

Dotychczas korzystaliśmy z pliku Dockerfile w celu zbudowania prostego obrazu z zainstalowaną aplikacją NGINX. Czas zbudować coś bardziej dynamicznego. Skorzystamy z systemu Alpine Linux i wykonamy następujące operacje:

- Stworzymy zmienną środowiskową określającą wersję PHP, którą chcemy zainstalować.
- Zainstalujemy serwer Apache2 i wybraną wersję PHP.
- Skonfigurujemy obraz tak, aby serwer Apache2 uruchamiał się bezproblemowo.
- Usuniemy domyślny plik *index.html* i dodamy plik *index.php*, który będzie wyświetlać wyniki wywołania polecenia *phpinfo*.
- Odsłonimy port kontenera o numerze 80.
- Skonfigurujemy serwer Apache tak, aby uruchamiał się jako domyślny proces.

Nasz plik Dockerfile zawiera następujące instrukcje:

```
FROM alpine:latest
LABEL maintainer="Russ Kendrick <russ@mckendrick.io>"
LABEL description="Plik demonstracyjny - instalacja Apache i PHP."
ENV PHPVERSION 7

RUN apk add --update apache2 php${PHPVERSION}-apache2 php${PHPVERSION} && \
    rm -rf /var/cache/apk/* && \
    mkdir /run/apache2/ && \
    rm -rf /var/www/localhost/htdocs/index.html && \
    echo "<?php phpinfo(); ?>" > /var/www/localhost/htdocs/index.php && \
    chmod 755 /var/www/localhost/htdocs/index.php

EXPOSE 80/tcp

ENTRYPOINT ["httpd"]
CMD ["-D", "FOREGROUND"]
```

Jak widzisz, zdecydowaliśmy się na zainstalowanie PHP7. Proces budowy obrazu należy w związku z tym uruchomić poleceniem:

```
$ docker build --tag local/apache-php:7 .
```

Zauważ, że polecenie to zostało nieco zmodyfikowane: tym razem obrazowi nadaliśmy nazwę *local/apache-php* i etykietę w postaci numeru wersji (7) (patrz pierwszy zrzut na następnej stronie).

Aby sprawdzić, czy wszystko przebiegło poprawnie, uruchom następujące polecenie budujące kontener na bazie obrazu:

```
$ docker container run -d -p 8080:80 --name apache-php7 local/apache-php:7
```

Po uruchomieniu kontenera w oknie przeglądarki wpisz adres **<http://localhost:8080/>**. Powinno to spowodować załadowanie strony z informacjami o zainstalowanej wersji PHP7 (patrz drugi zrzut na następnej stronie).

```
km@km-Extensa-5220: ~/kod/r02/env-test
km@km-Extensa-5220:~/kod/r02/env-test$ sudo docker build --tag local/apache-php:
7 .
[sudo] hasło użytkownika km:
Sending build context to Docker daemon 2.56kB
Step 1/8 : FROM alpine:latest
--> 3fd9065eaf02
Step 2/8 : LABEL maintainer="Russ McKendrick <russ@mckendrick.io>"
--> Using cache
--> 07923e5a70fe
Step 3/8 : LABEL description="Plik demonstracyjny - instalacja Apache i PHP."
--> Using cache
--> 6028438cd390
Step 4/8 : ENV PHPVERSION 7
--> Using cache
--> 29b5a96158f5
Step 5/8 : RUN apk add --update apache2 php${PHPVERSION}-apache2 php${PHPVERSION}
} &&
rm -rf /var/cache/apk/* && mkdir /run/apache2/ && rm -rf /var/www/localhost/htdocs/index.html && echo "<?php phpinfo(); ?>" > /var/www/localhost/htdocs/index.php && chmod 755 /var/www/localhost/htdocs/index.php
--> Using cache
--> da3d81a3f221
Step 6/8 : EXPOSE 80/tcp
--> Using cache
--> c6874496b3e7
Step 7/8 : ENTRYPOINT ["httpd"]
--> Using cache
--> ff523af578b8
Step 8/8 : CMD ["-D", "FOREGROUND"]
--> Using cache
--> 63a984db23eb
Successfully built 63a984db23eb
Successfully tagged local/apache-php:7
km@km-Extensa-5220:~/kod/r02/env-test$
```

phpinfo() - Mozilla Firefox

| System | |
|--|---|
| | Linux 95f1bfbf37aa 4.10.0-33-generic #37~16.04.1-Ubuntu SMP Fri Aug 11 14:07:24 UTC 2017 x86_64 |
| Build Date | |
| | Nov 28 2017 20:05:58 |
| Configure Command | |
| './configure' --build=x86_64-alpine-linux-musl' '--host=x86_64-alpine-linux-musl' '--prefix=/usr' '--program-suffix=-7' '--with-user=www-data' '--with-dir=/var/www/html' '--sysconfdir=/etc/php/7/' '--localstatedir=/var' '--with-lzma=/usr/lib' '--with-pcre=/usr/include/pcre' '--with-config-file-path=/etc/php/7/' '--with-config-file-scan-dir=/etc/php/7/conf.d' '--disable-short-tags' '--enable-bcmath=shared' '--with-bz2=shared' '--enable-calendar=shared' '--enable-type=shared' '--with-curl=shared' '--enable-dba=shared' '--with-db4' '--with-dbmaker=shared' '--with-gdbm' '--enable-dom=shared' '--enable-ftp=shared' '--enable-fpm=shared' '--enable-geoip=shared' '--enable-ftp=shared' '--with-xml=shared' '--with-freetype-dir=/usr' '--enable-gd-jis-conv=shared' '--enable-gd-native-ttf=shared' '--with-peg-dir=/usr' '--with-png-dir=/usr' '--with-webp-dir=/usr' '--with-xpm-dir=/usr' '--with-gettext=shared' '--with-gmp=shared' '--with-iconv=shared' '--with-imap=shared' '--with-imap-ssl' '--with-icu-dir=/usr' '--enable-intl=shared' '--enable-json=shared' '--with-kerberos' '--with-ldap=shared' '--with-libxml-dir=/usr' '--enable-libxml=shared' '--with-mcrypt=shared' '--enable-mbstring=shared' '--with-mysqli=shared' '--with-mysqli=shared' '--enable-mysqli=shared' '--with-mysqli=shared' '--enable-mysqli=shared' '--enable-openssl=shared' '--with-system-ciphers' '--enable-pcntl=shared' '--with-pcre=regex=usr' '--enable-pdo=shared' '--with-pdo-iblib=shared' '--with-pdo-mysql=shared' 'mysqlnd' '--with-pdo-odbc=shared' 'unixODBC' '--with-pdo-pgsql=shared' '--with-pdo-pspell=shared' '--enable-phar=shared' '--with-pspell=shared' '--with-pspell=shared' '--enable-session=shared' '--enable-shmop=shared' '--enable-simplexml=shared' '--with-smtp=shared' '--enable-soap=shared' '--enable-sockets=shared' '--with-sqlite3=shared' '--enable-sysvmsg=shared' '--enable-sysvsem=shared' '--enable-sysvshm=shared' '--with-tidy=shared' '--enable-tokenizer=shared' '--with-unixODBC=shared' '--enable-wddx=shared' '--enable-xsl=shared' '--enable-zip=shared' '--with-bz2=shared' '--enable-bz2=shared' '--enable-cgi=shared' '--with-apxs2' '--build_alias=x86_64-alpine-linux-musl' '--host_alias=x86_64-alpine-linux-musl' | |
| Server API | |
| Apache 2.0 Handler | |
| Virtual Directory Support | |
| disabled | |
| Configuration File (php.ini) Path | |
| /etc/php7 | |
| Loaded Configuration File | |
| /etc/php7/php.ini | |
| Scan this dir for additional .ini files | |
| /etc/php7/conf.d | |
| Additional .ini files parsed | |
| (none) | |
| PHP API | |
| 20160303 | |
| PHP Extension | |
| 20160303 | |

Podczas lektury dalszej części tej sekcji pamiętaj o tym, że nie ma czegoś takiego jak PHP6. Więcej informacji na ten temat znajdziesz na stronie <https://wiki.php.net/rfc/php6>.

Teraz zmodyfikuj swój plik Dockerfile — zmień wartość PHPVERSION z 7 na 5, a następnie uruchom poniższe polecenie w celu zbudowania nowego obrazu:

```
$ docker image build --tag local/apache-php:5 .
```

Jeżeli przyjrzyisz się komunikatom wyświetlonym w oknie Terminala, to zauważysz, że większość z nich jest taka sama jak wcześniej, ale instalowane są inne pakiety:

```
km@km-Extensa-5220:~/kod/r02/env-test$ sudo docker image build --tag local/apache-php:5 .
Sending build context to Docker daemon    2.56kB
Step 1/8 : FROM alpine:latest
--> 3fd9065eaf02
Step 2/8 : LABEL maintainer="Russ Kendrick <russ@mckendrick.io>"
--> Using cache
--> 07923e5a70fe
Step 3/8 : LABEL description="Plik demonstracyjny - instalacja Apache i PHP."
--> Using cache
--> 6028438cd390
Step 4/8 : ENV PHPVERSION 5
--> Using cache
--> c0f90ecfa827
Step 5/8 : RUN apk add --update apache2 php${PHPVERSION}-apache2 php${PHPVERSION}
} &&
rm -rf /var/cache/apk/* && mkdir /run/apache2/ &&
r m -rf /var/www/localhost/htdocs/index.html && echo "<?php phpinfo(); ?>" > /var/www/localhost/htdocs/index.php && chmod 755 /var/www/localhost/htdocs/index.php
--> Using cache
--> d082201638aa
Step 6/8 : EXPOSE 80/tcp
--> Using cache
--> 7cdc891a39b7
Step 7/8 : ENTRYPOINT ["httpd"]
--> Using cache
--> 969fa6361323
Step 8/8 : CMD ["-D", "BACKGROUND"]
--> Using cache
--> 41f99b98d9fc
Successfully built 41f99b98d9fc
Successfully tagged local/apache-php:5
km@km-Extensa-5220:~/kod/r02/env-test$
```

Spróbujmy uruchomić ten kontener. Tym razem skorzystajmy z portu 9090:

```
$ docker container run -d -p 9090:80 --name apache-php5 local/apache-php:5
```

Otwórz ponownie przeglądarkę, ale tym razem wejdź na stronę <http://localhost:9090/>. Zobaczysz informacje o zainstalowanej wersji PHP5:

The screenshot shows the output of the `phpinfo()` function in a Mozilla Firefox browser. The title bar indicates the page is "phpinfo() - Mozilla Firefox". The address bar shows "localhost:9090". The main content area is titled "PHP Version 5.6.30" and contains a large table with detailed PHP configuration settings. Key sections include "System", "Build Date", "Configure Command", "Server API", "Virtual Directory Support", "Configuration File (php.ini) Path", and "Loaded Configuration File". The "Configure Command" section is particularly long, listing numerous command-line options used to build the PHP extension.

Teraz możesz porównać rozmiary obrazów. Uruchom polecenie:

```
$ docker image ls
```

W oknie Terminala wyświetlane zostaną następujące informacje:

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|------------------|--------|--------------|----------------|---------|
| local/apache-php | 5 | de3cbc1c775b | 17 minutes ago | 43.9 MB |
| local/apache-php | 7 | b15a887563e2 | 22 minutes ago | 20.1 MB |
| alpine | latest | a41a7446062d | 2 weeks ago | 3.97 MB |

Jak widzisz, obraz PHP7 jest o wiele mniejszy od obrazu PHP5. Sprawdźmy, co tak naprawdę się stało.

Do czego doszło? Docker uruchomił obraz Alpine Linux w celu utworzenia obrazu. Na początku Docker zdefiniował określone przez nas zmienne środowiskowe, dzięki czemu zostały one udostępnione wszystkim procesom powłoki wewnętrz kontenera.

Na szczęście zasada nadawania nazw PHP w systemie Alpine Linux umożliwia prostą zmianę liczby w celu pobrania innej wersji PHP — nazwy pakietów, które muszą być zainstalowane, są identyczne, a więc uruchamiamy polecenie:

```
RUN apk add --update apache2 php${PHPVERSION}-apache2 php${PHPVERSION}
```

Zostało ono zinterpretowane w następujący sposób:

```
RUN apk add --update apache2 php7-apache2 php7
```

A w przypadku PHP5 przyjęto następującą interpretację:

```
RUN apk add --update apache2 php5-apache2 php5
```

Dzięki temu nie musimy samodzielnie przeglądać całego pliku Dockerfile i ręcznie zmieniać wszystkich numerów wersji. Rozwiążanie to przydaje się szczególnie podczas instalowania pakietów ze zdalnych adresów. Teraz przyjrzyjmy się plikowi Docker instalującemu i konfiguруjącemu aplikację Consul opracowaną przez HashiCorp. Narzędziu temu przyjrzymy się bliżej w ostatnim rozdziale tej książki. W zaprezentowanym przykładzie korzystamy ze zmienionych środowiskowych w celu zdefiniowania numerów wersji i sumy kontrolnej SHA256 obu pobieranych plików:

```
FROM alpine:latest
LABEL maintainer="Russ Kendrick <russ@mckendrick.io>"
LABEL description="Obraz z najnowszą wersją aplikacji Consul."

ENV CONSUL_VERSION 0.8.4
ENV CONSUL_SHA256 c8859a0a34c50115cdff147f998b2b63226f5f052e50f342209142420d1c2668
ENV CONSUL_UI_SHA256 7a49924a872205002b2bf72af8c82d5560d4a7f4a58b2f65ee284dd254ebd063

RUN apk add --update ca-certificates wget && \
    wget -O consul.zip \
    https://releases.hashicorp.com/consul/${CONSUL_VERSION}/consul_${CONSUL_VERSION}_ \
    →linux_amd64.zip && \
    echo "$CONSUL_SHA256 *consul.zip" | sha256sum -c - && \
    unzip consul.zip && \
    mv consul /bin/ && \
    rm -rf consul.zip && \
    cd /tmp && \
    wget -O ui.zip \
    https://releases.hashicorp.com/consul/${CONSUL_VERSION}/consul_${CONSUL_VERSION}_web_ \
    →ui.zip && \
    echo "$CONSUL_UI_SHA256 *ui.zip" | sha256sum -c - && \
    unzip ui.zip && \
    mkdir -p /ui && \
    mv * /ui && \
    rm -rf /tmp/* /var/cache/apk/*

EXPOSE 8300 8301 8301/udp 8302 8302/udp 8400 8500 8600 8600/udp

VOLUME [ "/data" ]

ENTRYPOINT [ "/bin/consul" ]
CMD [ "agent", "-data-dir", "/data", "-server", "-bootstrap-expect", "1", "-ui-dir", \
    →"/ui", "-client=0.0.0.0" ]
```

Jak widzisz, zawartość pliku Dockerfile może być rozbudowana, a zmienne środowiskowe pomagają w jego utrzymaniu. W przyszłości, gdy pojawi się nowa wersja aplikacji Consul, można po prostu zmienić trzy linie instrukcji ENV, co przy odpowiedniej konfiguracji spowodowałoby zbudowanie nowego obrazu, ale zagadnieniu temu przyjrzymy się dopiero w kolejnym rozdziale.

W przedstawionym pliku Dockerfile znajduje się polecenie VOLUME, które nie zostało jeszcze opisane. Nie martw się. Więcej informacji na jego temat znajdziesz w rozdziale 4. „Zarządzanie kontenerami”.

Podsumowanie

W tym rozdziale przyjrzaliśmy się plikom Dockerfile, opisaliśmy dobre praktyki pracy nad nimi, przedstawiliśmy polecenie docker image build, a także różne sposoby budowania kontenerów. Podczas lektury tego rozdziału poznaleś zmienne środowiskowe i dowiedziałeś się, jak można ich używać w celu przekazania różnych parametrów z pliku Dockerfile do kontenera.

Wiesz już, jak budować obrazy za pomocą plików Dockerfile, a więc w kolejnym rozdziale nauczysz się korzystać z repozytorium Docker Hub i poznasz zalety użytkowania tego rejestru. Ponadto przyjrzymy się rejestrowi Dockera. Jest to otwarte oprogramowanie umożliwiające tworzenie własnych miejsc do przechowywania obrazów bez konieczności wnoszenia opłat za korzystanie z zewnętrznej chmury Docker Enterprise Cloud.

Przechowywanie obrazów i ich dystrybucja

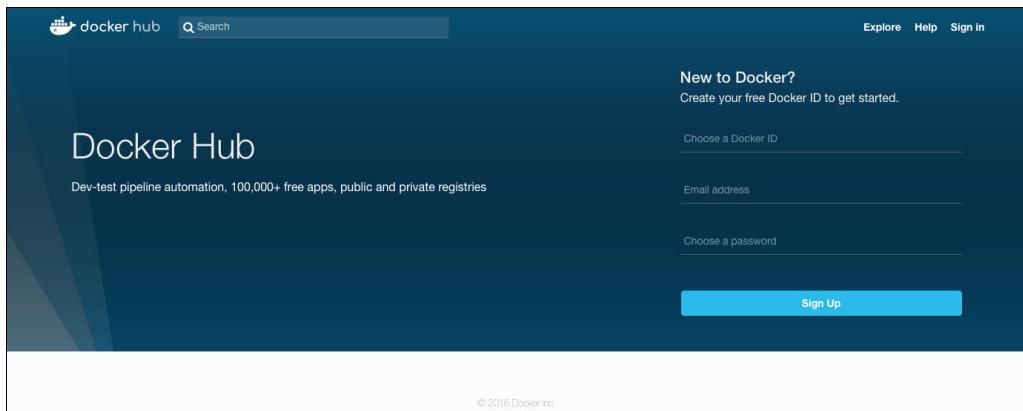
W tym rozdziale opiszemy możliwości przechowywania i udostępniania obrazów oferowane między innymi przez serwis Docker Hub i rejestr Docker Registry, który może być również uruchomiony w środowisku lokalnym. Przedstawimy różnice pomiędzy tymi usługami i ich zastosowania. W rozdziale tym również zajmiemy się automatyzacją budowy obrazów za pomocą narzędzi sieciowych. Oto lista tematów, które poruszymy:

- Repozytorium Docker Hub,
- Serwis Docker Store,
- rejestr Docker Registry,
- niezależne rejesty,
- narzędzie Microbadge.

Repozytorium Docker Hub

O repozytorium Docker Hub pisaliśmy w dwóch poprzednich rozdziałach, ale korzystaliśmy z niego tylko do pobierania obrazów za pomocą polecenia `docker image pull`. Docker Hub umożliwia darmowe upublicznanie obrazów, ale również oferuje płatną usługę tworzenia prywatnych repozytoriów. Opiszemy działanie serwisu Docker Hub i dostępnych w nim opcji umożliwiających zarządzanie obrazami kontenerów.

Na poniższym zrzucie pokazano stronę główną serwisu Docker Hub znajdującą się pod adresem <https://hub.docker.com>.



W prawym górnym rogu strony znajduje się formularz pozwalający na założenie konta w serwisie Docker Hub. Jeżeli korzystałeś wcześniej z Dockera, to prawdopodobnie masz już konto w serwisie Docker Hub. Jeżeli jeszcze go nie założyłeś, to zrób to za pomocą formularza widocznego po prawej stronie okna (na koniec kliknij przycisk *Sign Up*). Jeżeli masz już konto w serwisie Docker Hub, to się po prostu do niego zaloguj, korzystając z opcji *Sign In*.

Korzystanie z serwisu Docker Hub jest darmowe i jeżeli nie chcesz ładować do niego własnych obrazów ani nimi zarządzać, to do samego pobierania obrazów nie musisz zakładać konta użytkownika Docker Hub.

Panel główny

Po zalogowaniu się do serwisu Docker Hub zobaczysz poniższą stronę. Jest to **ekran główny** serwisu Docker Hub (tzw. *dashboard*) (patrz pierwszy zrzut na następnej stronie).

Stąd możesz wejść na wszystkie podstrony serwisu Docker Hub. Zanim przejdziemy do nich, chcielibyśmy zająć się ekranem głównym. Wyświetla on listę wszystkich Twoich obrazów (obrazów publicznych i prywatnych). Są one sortowane według liczby gwiazdek (oceny) i liczby pobrań. W momencie pisania tej książki nie można zmienić tego domyślnego sposobu sortowania.

W kolejnych sekcjach opiszymy wszystkie elementy ekranu głównego. Zaczniemy od niebieskiej belki znajdującej się u góry ekranu.

Type to filter repositories by name

| | | | |
|--|---------|------------|-------------------------|
|  russmckendrick/ab public automated build | 1 STARS | 4.0K PULLS | DETAILS |
|  russmckendrick/cluster public automated build | 1 STARS | 1.6K PULLS | DETAILS |
|  russmckendrick/base public automated build | 1 STARS | 1.3K PULLS | DETAILS |
|  russmckendrick/nginx-php public automated build | 1 STARS | 1.1K PULLS | DETAILS |
|  russmckendrick/consul public automated build | 0 STARS | 854 PULLS | DETAILS |

Przycisk Explore

Przycisk *Explore* (eksploruj) wyświetla listę oficjalnych obrazów Dockera. Lista ta, podobnie jak lista Twoich obrazów, jest posortowana według oceny i liczby pobrań. Jak widzisz na powyższym zrzucie, każdy z oficjalnych obrazów został pobrany ponad 10 milionów razy:

Docker Store is the new place to discover public Docker content. Check it out →

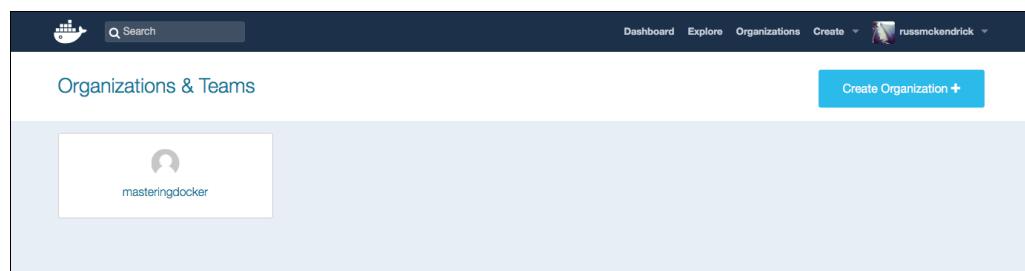
Explore Official Repositories

| | | | |
|---|------------|------------|-------------------------|
|  nginx official | 6.2K STARS | 10M+ PULLS | DETAILS |
|  redis official | 3.9K STARS | 10M+ PULLS | DETAILS |
|  busybox official | 1.0K STARS | 10M+ PULLS | DETAILS |
|  ubuntu official | 6.1K STARS | 10M+ PULLS | DETAILS |
|  docker official | 1.5K STARS | 10M+ PULLS | DETAILS |

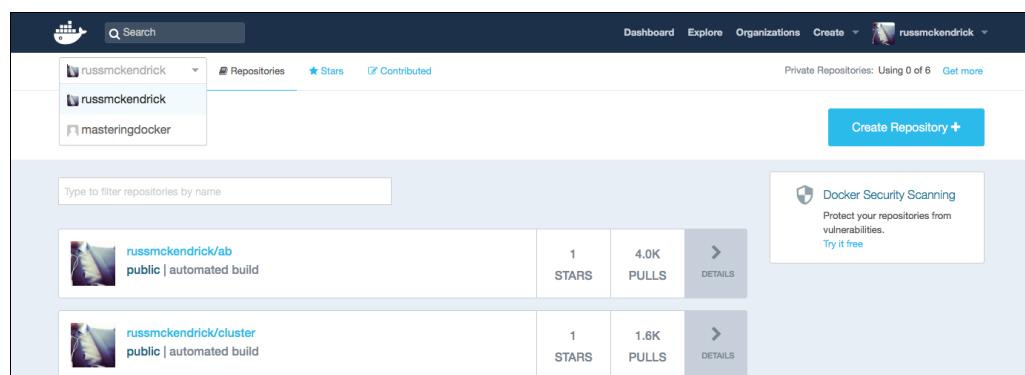
W górnej części strony wyświetlana jest reklama zachęcająca do wypróbowania sklepu Docker Store. Serwisowi temu przyjrzymy się w dalszej części tego rozdziału.

Przycisk Organizations

Przycisk *Organizations* (organizacje) wyświetla utworzone przez Ciebie organizacje, a także organizacje, do których Cię dodano. Organizacje umożliwiają wspólną pracę kilku osób nad projektem. Dysponują one własnymi ustawieniami określającymi np. to, czy używane przez nie repozytorium jest publiczne, czy prywatne — Docker Hub umożliwia tworzenie niezależnych prywatnych repozytoriów przypisanych do organizacji.

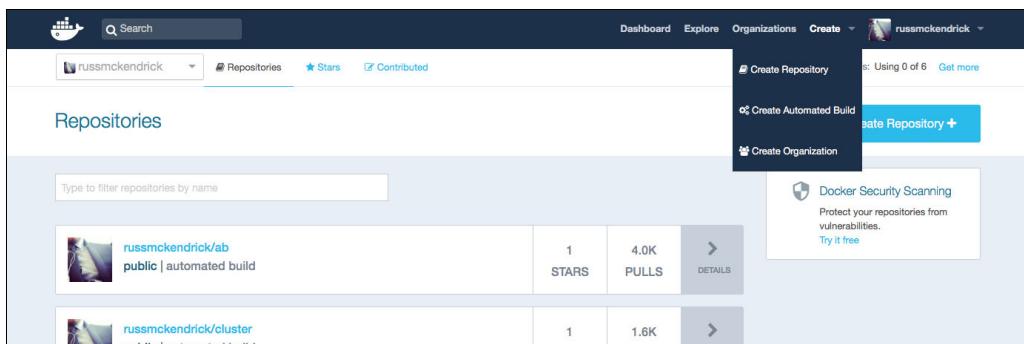


Na panelu głównym, pod logo Dockera, znajduje się przycisk umożliwiający przełączanie się pomiędzy kontami i organizacjami (zwykle wyświetlana jest tam nazwa zalogowanego użytkownika):



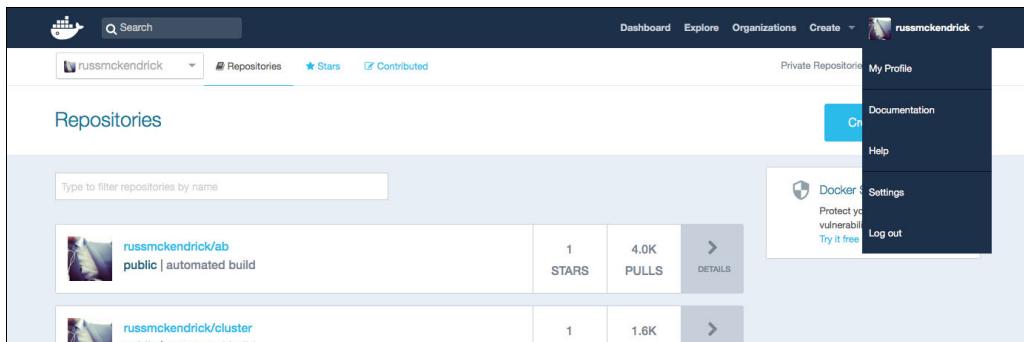
Menu Create

Menu *Create* (utwórz) umożliwia tworzenie repozytoriów i automatyzację procesu budowy obrazów. Zagadnienia te opiszemy w dalszej części tego rozdziału. Po kliknięciu przycisku *Create* zobaczysz menu zawierające trzy opcje:



Profil i ustawienia

Ostatnie menu górnej belki służy do zarządzania profilem (*My Profile*) i ustawieniami (*Settings*):



Strona *Settings* (ustawienia) pozwala na konfigurację profilu publicznego:

- zmianę hasła;
- przejrzenie listy organizacji, do których należysz;
- przejrzenie subskrypcji dostarczanych na Twój adres e-mail;
- zdefiniowanie powiadomień, które chcesz otrzymywać;
- zdefiniowanie autoryzowanych usług, które mogą uzyskiwać dostęp do Twoich danych;
- przejrzenie dołączonych kont (np. kont w serwisach GitHub lub Bitbucket);
- zapoznanie się z licencją usług komercyjnych, rachunkami i ustawieniami globalnymi.

Obecnie jedyną globalną opcją jest możliwość wyboru tego, czy nowo utworzone repozytoria mają być domyślnie publiczne, czy prywatne. Standardowo przyjęta jest opcja ich upublicznienia.

The screenshot shows the 'Account Settings' section of the Docker Hub account. It includes fields for setting repository visibility (public or private) and managing email addresses (adding a new one or marking an existing one as primary).

Default Repository Visibility
Update the default visibility for your repositories.

public private

Email Addresses
This email address will be used for all notifications and correspondence from Docker.

If you wish to designate a different email address as primary, first add a new address to your account and then click "make primary".

New Email

| | | |
|--------------------|----------|---------|
| russ@mckendrick.io | verified | primary |
|--------------------|----------|---------|

Opcja *My Profile* (mój profil) wyświetla stronę Twojego profilu publicznego. W celu wejścia na mój profil skorzystaj z adresu <https://hub.docker.com/u/russmckendrick/>. Obecnie wygląda on tak:

The screenshot shows the Docker Hub profile page for 'russmckendrick'. On the left, there's a large profile picture and the user's name 'russmckendrick' followed by 'Russ McKendrick'. Below this, there are profile details: 'Nottingham, UK', a link to 'https://media-glass.es/', and the date 'Joined February 2014'. To the right, there's a list of repositories under the heading 'Stars'. Each repository entry includes the name, visibility (public), description ('automated build'), stars (1), pulls (4.0K), and a 'DETAILS' button.

| Repository | Description | Stars | Pulls | Action |
|--------------------------|--------------------------|---------|------------|-------------------------|
| russmckendrick/ab | public automated build | 1 STARS | 4.0K PULLS | DETAILS |
| russmckendrick/cluster | public automated build | 1 STARS | 1.6K PULLS | DETAILS |
| russmckendrick/base | public automated build | 1 STARS | 1.3K PULLS | DETAILS |
| russmckendrick/nginx-php | public automated build | 1 STARS | 1.1K PULLS | DETAILS |
| russmckendrick/consul | public automated build | 0 STARS | 854 PULLS | DETAILS |
| russmckendrick/mariadb | public automated build | 1 STARS | 850 PULLS | DETAILS |

Strona Stars

Na głównym panelu, pod czarną górną belką, znajdują się jeszcze dwie rzeczy, których nie opisaliśmy. Pierwszą z nich jest przycisk otwierający stronę *Stars* (gwiazdy) pozwalającą na przejrzenie listy repozytoriów, które oznaczyłeś wcześniej jako ulubione (patrz pierwszy zrzut na następnej stronie).

Do listy ulubionych warto dodawać repozytoria, z których lubisz korzystać. Dzięki tej liście będziesz mógł z łatwością śledzić udostępniane na nich nowości i aktualizacje (patrz drugi zrzut na następnej stronie).

The screenshot shows the Jenkins GitHub organization page. At the top, there's a search bar with 'jenkins' and a dropdown for 'russmckendrick'. Below the search bar, there are tabs for 'Dashboard', 'Explore', 'Organizations', 'Create', and a dropdown for 'russmckendrick'. Under the 'Repositories' tab, there are three repository cards:

- alpine** official: 2.3K STARS, 10M+ PULLS, DETAILS
- jenkins** official: 2.9K STARS, 10M+ PULLS, DETAILS
- nginx** official: 6.2K STARS, 10M+ PULLS, DETAILS

At the bottom right of the card area, it says 'Private Repositories: Using 0 of 6 Get more'.

Drugim elementem jest przycisk *Contributed* (udział) wyświetlający listę repozytoriów, w których rozwój wniosłeś własny wkład, a które nie znajdują się na liście *Repositories*.

Automatyzacja budowy obrazu

W tej sekcji przyjrzymy się automatyzacji budowy obrazu. Docker Hub umożliwia połaczenie z kontami w repozytoriach GitHub i Bitbucket w celu automatycznego przebudowania obrazu po wprowadzeniu zmian w kodzie umieszczonego w tych repozytoriach. W kolejnych sekcjach przyjrzymy się wszystkim elementom procesu automatyzacji.

Umieszczanie kodu w repozytorium

Pracę nad automatyzacją procesu budowy obrazów należy rozpocząć od założenia repozytorium GitHub lub Bitbucket. Wybierając miejsce przechowywania kodu, możesz skorzystać z tych dwóch opcji. Konfiguracja będzie wyglądała tak samo w obu przypadkach, a w dalszej części tej książki będę korzystał z repozytorium dostępnego pod adresem <https://github.com/russmckendrick/mastering-docker/>.

Moje repozytorium jest ogólnodostępne, a więc możesz skopiować jego zawartość lub pobrać spoolonizowane wersje kodu ze strony <ftp://ftp.helion.pl/przyklady/dockaz.zip> i załadować kod do własnego repozytorium, co pozwoli Ci na samodzielne wykonywanie opisanych przeze mnie czynności:

The screenshot shows the GitHub repository page for 'russmckendrick / mastering-docker'. At the top, there are buttons for 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Settings', and 'Insights'. Below that, there's a dropdown for 'Branch: master' and a path 'mastering-docker / chapter02 / dockerfile-example /'. There are buttons for 'Create new file', 'Upload files', 'Find file', and 'History'. The main area shows a commit history:

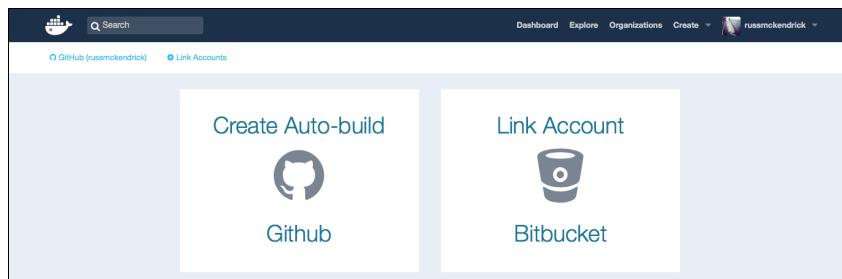
| Author | Message | Date |
|----------------|---------------------------|----------------------------------|
| russmckendrick | Adding Dockerfile example | Latest commit 7be30cd 7 days ago |
| .. | | |
| files | Adding Dockerfile example | 7 days ago |
| Dockerfile | Adding Dockerfile example | 7 days ago |

At the bottom, there are links for 'Contact GitHub API Training Shop Blog About'.

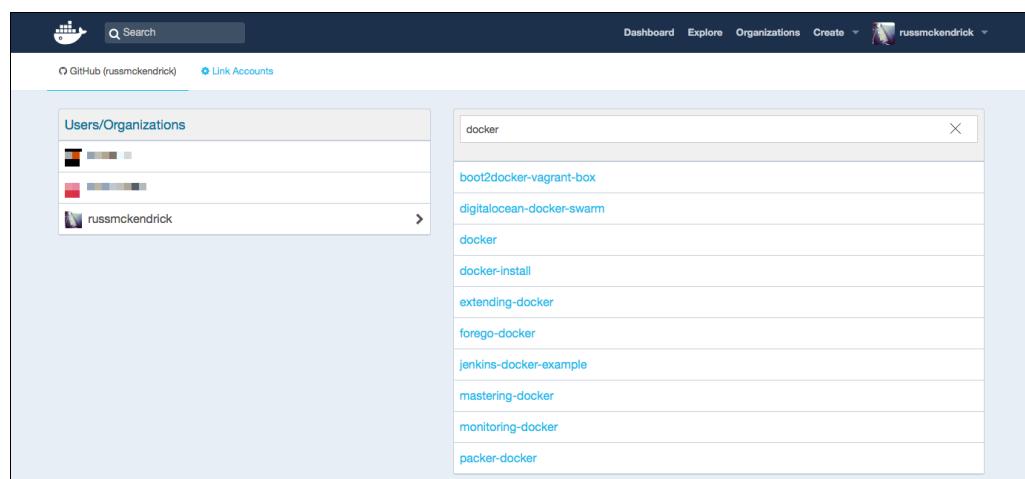
W rozdziale 2. „Tworzenie obrazów kontenerów” opisaliśmy przykładowy plik Dockerfile. Będziemy z niego korzystać ponownie. Przypominamy, że w obrazie tym zainstalowaliśmy aplikację NGNIX i dodaliśmy do niego prostą stronę zawierającą komunikat powitalny.

Konfiguracja serwisu Docker Hub

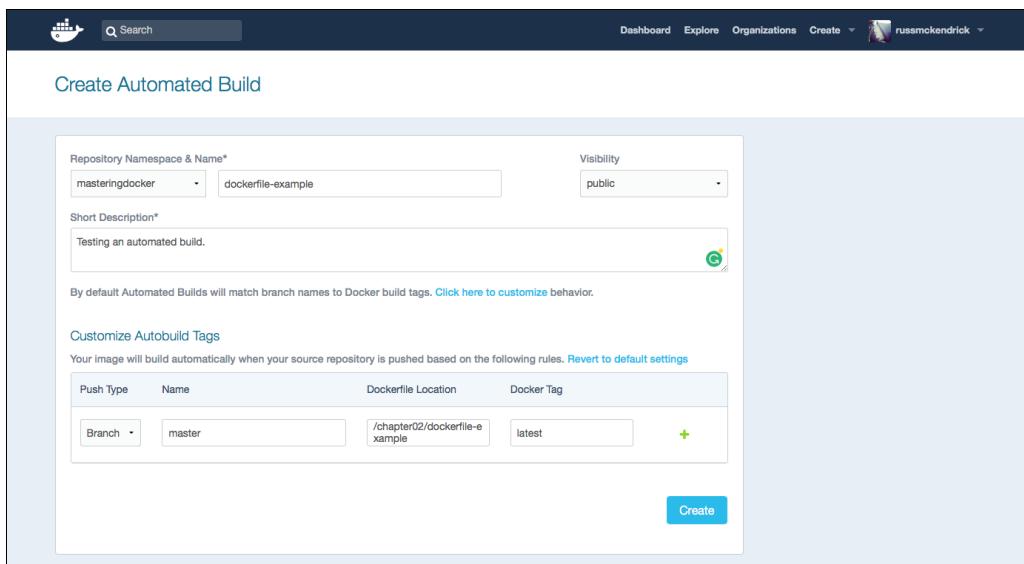
Otwórz panel główny serwisu Docker Hub, rozwiń menu *Create* (utwórz) i wybierz z niego opcję *Create Automated Build* (utwórz zautomatyzowany proces budowy obrazu). Otwarta zostanie strona zawierająca podłączone konta GitHub lub Bitbucket:



Musisz znaleźć i wybrać repozytorium GitHub lub Bitbucket, z którego chcesz brać dane służące do automatycznego budowania obrazów. Wygenerujesz w ten sposób mechanizm, który po zatwierdzeniu zmian w kodzie umieszczonym w repozytorium wywoła przebudowę obrazu udostępnianego w serwisie Docker Hub.



Wybrałem repozytorium o nazwie *mastering-docker* i wszedłem na stronę ustawień automatyzacji budowy obrazu. Na stronie tej można wybrać profil Docker Hub, do którego ma należeć obraz, nadać mu nazwę, a także określić jego dostępność i opis. Ponadto, klikając przycisk *Click here to customize* (kliknij tu, aby spersonalizować), możesz przeprowadzić personalizację ustawień. Serwis Docker Hub należy powiadomić o tym, gdzie znajduje się plik Dockerfile.



Oto dane, które wprowadziłem w poszczególnych polach podczas konfiguracji:

- *Repository Namespace & Name* (przestrzeń nazw i nazwa repozytorium): dockerfile-example (przykładowy plik Dockerfile);
- *Visibility* (widoczność): public (publiczny);
- *Short Description* (skrócony opis): Testing an automated build (testowanie procesu automatycznej budowy obrazów);
- *Push Type* (typ operacji wstawiania): Branch (gałąź);
- *Name* (nazwa): Master (główny);
- *Dockerfile Location* (położenie pliku Dockerfile): /chapter02/dockerfile-example/ (jeżeli korzystasz ze spoolonizowanych wersji kodu, to plik może znajdować się w folderze /r02/dockerfile/);
- *Docker Tag* (znacznik Dockera): latest (najnowszy).

Po kliknięciu przycisku *Create* (utwórz) na ekranie wyświetlona zostanie strona podobna do następującej (patrz pierwszy zrzut na następnej stronie).

Po zdefiniowaniu procesu budowy obrazu możemy przeprowadzić konfigurację dodatkowych opcji — kliknij *Build Settings* (ustawienia budowy obrazu). Korzystamy z oficjalnego obrazu Alpine Linux, a więc możemy go podłączyć pod proces automatycznego budowania naszego obrazu. W tym celu w sekcji *Repository Links* (adresy repozytoriów) wpisz alpine, a następnie kliknij przycisk *Add Repository Link* (dodaj adres repozytorium). Dzięki temu obraz będzie przebudowywany po każdej publikacji nowej wersji obrazu Alpine Linux (patrz drugi zrzut na następnej stronie).

Łączenie automatyzacji własnych obrazów z oficjalnymi obrazami lub własnymi obrazami bazowymi jest zalecane, ponieważ pozwala na uwzględnianie wszystkich aktualizacji i poprawek.

Docker. Programowanie aplikacji dla zaawansowanych

This screenshot shows the Docker Hub repository page for 'masteringdocker/dockerfile-example'. The repository is public and has never been pushed. It includes tabs for Repo Info, Tags, Dockerfile, Build Details, Build Settings, Collaborators, Webhooks, and Settings. The Repo Info section contains fields for Short Description (Testing an automated build.), Docker Pull Command (docker pull masteringdocker/dockerfile-example), Full Description (Full description is empty for this repo.), Owner (masteringdocker), and Source Repository (russmckendrick/mastering-docker).

This screenshot shows the Build Settings page for the same repository. It includes sections for Build Settings (checkbox for automatic builds on pushes), Build Rules (specifying Dockerfile location and tag name), Repository Links (linking to another repository to trigger rebuilds), and Linked Repositories (listing 'alpine'). A 'Save Changes' button is visible.

Od teraz nasz obraz będzie automatycznie przebudowywany i publikowany po każdej poprawce wprowadzonej w repozytorium GitHub i po każdej publikacji nowej wersji oficjalnego obrazu wybranego systemu operacyjnego. Do żadnego z tych zdarzeń prawdopodobnie nie dojdzie natychmiast, a więc kliknij przycisk *Trigger* (aktywuj) w celu ręcznej aktywacji procesu budowy obrazu.

Kliknięcie przycisku *Build Details* (szczegóły budowy) spowoduje wyświetlenie listy wszystkich procesów budowy obrazu (na liście znajdują się procesy wykonane poprawnie oraz te, podczas których wystąpiły błędy). Twój lista powinna zawierać informacje o wykonanym przed chwilą procesie budowy obrazu. Kliknięcie tego procesu spowoduje wyświetlenie jego dziennika:

The screenshot shows a GitHub automated build details page for a repository named `masteringdocker/dockerfile-example`. The page has a dark header with the GitHub logo, a search bar, and navigation links for Dashboard, Explore, Organizations, Create, and a user profile for `russmckendrick`.

The main content area is titled "PUBLIC | AUTOMATED BUILD" and displays the build log for the latest commit. The log shows:

- Last pushed:** a few seconds ago
- Repo Info:** master branch, Dockerfile location: `/chapter02/dockerfile-example/`
- Build Details:** Docker Tag: latest, Build Created: a few seconds ago, UTC: 2017-06-17T14:03:06.938Z
- Build Code:** A snippet of the Dockerfile code is shown, starting with `FROM alpine:latest` and ending with `CMD ["-g", "daemon off;"]`.
- README:** An empty file.
- Dockerfile:** The full Dockerfile code is displayed in a monospaced font.

Po zbudowaniu obrazu możesz go pobrać do swojego lokalnego środowiska Docker. Skorzystaj z następujących poleceń (pamiętaj o podaniu nazw własnego repozytorium i własnego obrazu):

```
$ docker pull masteringdocker/dockerfile-example
$ docker image ls
```

Oto komunikaty wyświetlane w oknie Terminala po uruchomieniu tych polecień:

```
russ in ~
⚡ docker pull masteringdocker/dockerfile-example
Using default tag: latest
latest: Pulling from masteringdocker/dockerfile-example
2aecc7e1714b: Pull complete
059b14890b0a1: Pull complete
6e4602f97fc3: Pull complete
ee5e31fa6ddd: Pull complete
4b30bc355ec2: Pull complete
Digest: sha256:a656ed03b14f726510e3a62cb072cdcc7e4cfeddb9f8e52ee694c93612a42bea
Status: Downloaded newer image for masteringdocker/dockerfile-example:latest
russ in ~
⚡ docker image ls
REPOSITORY           TAG      IMAGE ID      CREATED     SIZE
masteringdocker/dockerfile-example  latest    077956f5adfe   3 minutes ago  5.49 MB
russ in ~
⚡ 
```

W celu uruchomienia pobranego kontenera skorzystaj z poniższego polecenia (jeżeli pobrałeś własny obraz o innej nazwie, to pamiętaj o umieszczeniu w tym poleceniu nazwy własnego kontenera):

```
$ docker container run -d -p8080:80 masteringdocker/dockerfile-example
```

Ładowanie własnych obrazów

W rozdziale 2. „Tworzenie obrazów kontenerów” opisaliśmy zagadnienia związane z tworzeniem obrazu bez użycia pliku Dockerfile. Nie jest to najlepsza technika i staraj się z niej korzystać tylko w razie konieczności. Utworzony w ten sposób obraz może zostać również załadowany do serwisu Docker Hub.

Ładując w ten sposób obrazy do serwisu Docker Hub, upewnij się, że nie zawierają one kodu, plików lub zmiennych środowiskowych, których nie chciałbyś udostępnić innym.

W celu załadowania obrazu wykonanego samodzielnie należy najpierw połączyć lokalny klient Dockera z serwisem Docker Hub. W tym celu należy uruchomić następujące polecenie:

```
$ docker login
```

Teraz zostaniesz poproszony o podanie nazwy użytkownika (*Username*) i hasła (*Password*):

```
russ in ~
⚡ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: russmckendrick
Password:
Login Succeeded
russ in ~
⚡ 
```

Klient Dockera może już korzystać z serwisu Docker Hub. Do dalszej pracy potrzebujemy obrazu do zbudowania. Wróćmy do przykładu *scratch* opisanego w rozdziale 2. „Tworzenie obrazów kontenerów”. Zbudujmy ten obraz:

```
$ docker build --tag masteringdocker/scratch-test:latest .
```

Po uruchomieniu tego polecenia w oknie Terminala wyświetlane zostaną następujące komunikaty:

```
russ in ~/Documents/Code/mastering-docker/chapter02/scratch-example on master*
$ docker build --tag masteringdocker/scratch-example:latest .
Sending build context to Docker daemon 4.247 MB
Step 1/3 : FROM scratch
-->
Step 2/3 : ADD files/alpine-miniroots-3.6.1-x86_64.tar /
--> ceaf29dcf11d
Removing intermediate container 218ba6dde4b4
Step 3/3 : CMD /bin/sh
--> Running in 782c61a2f01f
--> 06f66706e5eb
Removing intermediate container 782c61a2f01f
Successfully built 06f66706e5eb
```

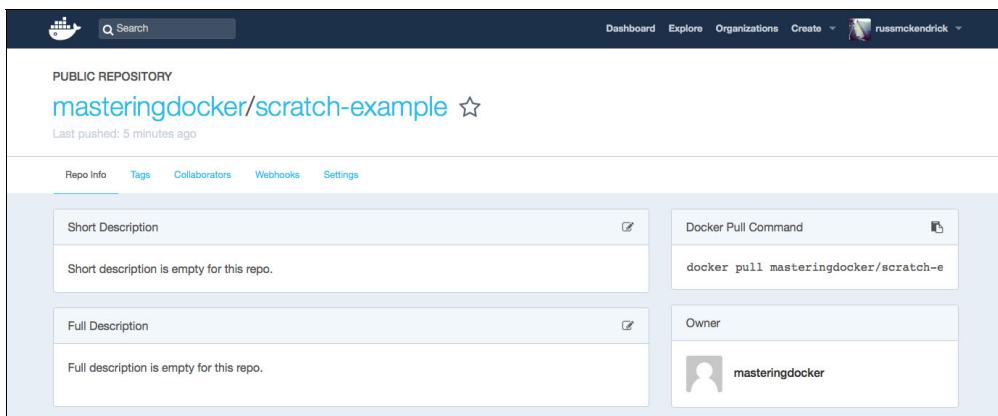
Po zbudowaniu obrazu możemy załadować go do serwisu Docker Hub za pomocą polecenia:

```
$ docker image push masteringdocker/scratc-testh:latest
```

Po uruchomieniu tego polecenia w oknie Terminala wyświetlane zostaną następujące komunikaty:

```
russ in ~/Documents/Code/mastering-docker/chapter02/scratch-example on master*
$ docker image push masteringdocker/scratch-example:latest
The push refers to a repository [docker.io/masteringdocker/scratch-example]
78268ba3e8f0: Pushed
latest: digest: sha256:5aba2e4761f4403beb110b13ca1c8ac62152045b93d086e5c0d1ee375f16e24e size: 528
russ in ~/Documents/Code/mastering-docker/chapter02/scratch-example on master*
```

Podczas budowy obraz oznaczamy etykietą `masteringdocker/scratch:latest`, a więc Docker automatycznie ładuje obraz do zdefiniowanego w ten sposób repozytorium, co z kolei spowodowało w moim przypadku dodanie nowego obrazu do organizacji *Mastering Docker* (mojemu obrazowi nadalem nazwę *scratch-example*):



Jeżeli przyjrzysz się dostępnym opcjom, to odkryjesz, że tak naprawdę nie możesz zrobić zbyt wiele z obrazem załadowanym do serwisu Docker Hub. Wynika to z tego, że obraz ten nie został zbudowany w serwisie Docker Hub, a więc serwis ten nie wie, jakie operacje zostały wykonane w celu zbudowania tego obrazu.

Serwis Docker Store

Być może pamiętasz, że w rozdziale 1. „Docker — wprowadzenie” pobieraliśmy Dockera w wersji dla systemu macOS i Windows z serwisu Docker Store. Docker Store pozwala na pobranie **Dockera CE** i **Dockera EE** w wersjach przystosowanych do pracy w różnych systemach operacyjnych. Ponadto serwis ten pozwala na pobieranie obrazów Dockera i **wtyczek**.

The screenshot shows the Docker Store interface with the following details:

- Filters:** Docker EE, Docker CE, **Containers** (selected), Plugins.
- 1 - 10 of 182 available images.**
- Most Popular** dropdown menu.
- Buildkite Pty Ltd** (Buildkite, Inc.):
 - Icon: Green square with white 'B'.
 - Description: The buildkite-agent is a small, reliable, and cross-platform build runner that makes it easy to run automate...
 - Tags: DevOps Tools.
- CoScale agent** (CoScale):
 - Icon: Yellow circle with 'C' and 'SCALE'.
 - Description: Production monitoring of Dockerized applications, powered by anomaly detection.
 - Tags: Monitoring, DevOps Tools, Analytics.
- Couchbase** (Couchbase Inc):
 - Icon: Red circle with white 'C'.
 - Description: Couchbase delivers the database for the Digital Economy. Developers around the world choose Couchbas...
 - Tags: Storage, Application Frameworks.

Na stronie serwisu Docker Store znajdziesz tylko oficjalne i certyfikowane obrazy, ale interfejs tego serwisu może być również użyty w celu przeszukiwania treści udostępnionych w repozytoriach Docker Hub. Ponadto umożliwia on pobieranie obrazów, które nie są dostępne w repozytoriach Docker Hub, takich jak np. obraz Citrix NetScaler CPX Express (<https://store.docker.com/images/netscaler-cpx-express/>) (patrz zrzut na następnej stronie).

Zauważ, że na stronie obrazu widoczna jest jego cena (wersja *Express* kosztuje 0 dolarów) — serwis Docker Store umożliwia zakup komercyjnego oprogramowania i dysponuje wbudowanym systemem obsługującym płatności i licencje. Jeżeli tworzysz oprogramowanie i chcesz sprzedawać je za pośrednictwem serwisu Docker Store, zajrzyj na stronę <https://store.docker.com/publisher/signup/>. Znajdziesz na niej więcej informacji na ten temat.

The screenshot shows a Docker Store page for the 'NetScaler CPX Express Rel 11.1' image. At the top, there's a navigation bar with 'Explore', 'Publish', 'Feedback', and 'Log In'. Below the header, the product title 'NetScaler CPX Express Rel 11.1' is displayed next to the Citrix logo. A brief description follows: 'Free, Full-featured, microservice aware, load balancer in a Docker container'. Underneath, it says 'By Citrix Systems' and lists 'Categories: Application Infrastructure, DevOps Tools'. To the right, there's a price section showing 'Express \$0.00' with a note about limited bandwidth and SSL connections. Below that is a 'Terms of Service' link and a 'Copy and paste to pull this image' button containing the command 'docker pull store/citrix/netscaler'. Further down, there are tabs for 'DESCRIPTION', 'REVIEWS', and 'RESOURCES'. A note states 'Release 11.1 is the stable build'. On the right side, there's a box prompting users to log in to write a review.

W kolejnych rozdziałach, w których będziemy opisywać wtyczki Dockera, wróćmy do zagadnień związanych z obsługą serwisu Docker Store.

Rejestr Docker Registry

W tej sekcji przyjrzymy się rejestrowi Docker Registry. Jest to otwarta aplikacja, którą możesz uruchomić w dowolnym środowisku i przechowywać w niej obrazy Dockera. Porównamy aplikacje Docker Registry i Docker Hub i pomożemy Ci wybrać tę, która lepiej sprawdzi się w Twojej pracy. Po przeczytaniu tej sekcji będziesz umiał uruchomić własny rejestr Docker Registry.

Docker Registry — informacje ogólne

Rejestr Docker Registry jest otwartą aplikacją przeznaczoną do przechowywania obrazów Dockera w wybranym środowisku. Pozwala ona na utrzymywanie ich we w pełni prywatnym systemie i udostępnianie w razie konieczności. Podstawowe informacje na temat tego rejestru znajdziesz na stronie <https://docs.docker.com/registry/>.

W tym podrozdziale chcielibyśmy przeprowadzić Cię przez proces konfiguracji aplikacji Docker Registry i publikowania w niej obrazów (porównamy ją z repozytorium Docker Hub). Aplikacja Docker Registry przydaje się w przypadku chęci wdrożenia własnego rejestru bez konieczności płacenia za prywatne funkcje serwisu Docker Hub. Porównamy funkcje Docker Hub i Docker Registry, co pozwoli Ci na podjęcie świadomej decyzji i wybranie właściwego miejsca przechowywania obrazów.

Dzięki Docker Registry będziesz mógł:

- Stworzyć własny rejestr, w którym będziesz mógł tworzyć prywatne i publiczne repozytoria, a następnie nim zarządzać.
- Skalować rejestr w zależności od liczby obrazów lub obciążenia operacjami pobierania obrazów.
- Obsługiwać wszystko z poziomu wiersza poleceń.

Docker Hub pozwala na:

- Zarządzanie obrazami za pomocą graficznego interfejsu użytkownika.
- Korzystanie z gotowego środowiska chmury obsługującego publiczne i prywatne obrazy.
- Nieprzejmowanie się problemami wynikającymi z zarządzania serwerem, na którym umieszczone są obrazy.

Wdrażanie własnego rejestru

Jak zapewne się domyślasz, aplikacja Docker Registry jest udostępniana w formie obrazu, który można wdrożyć za pomocą poniższych poleceń:

```
$ docker image pull registry:2  
$ docker container run -d -p 5000:5000 --name registry registry:2
```

Uruchomienie tych poleceń zainstaluje Docker Registry w sposób najbardziej podstawowy. Zobaczmy, jak można ładować obrazy do tego rejestru i jak można je z niego pobierać. Do eksperymentowania potrzebujemy jakiegoś obrazu. Pobierzmy ponownie obraz systemu Alpine Linux:

```
$ docker image pull alpine
```

Dysponujemy kopią obrazu Alpine Linux. Spróbujmy załadować go do lokalnego rejestru dostępnego pod adresem `localhost:5000`. W celu zafikowania obrazu do lokalnego rejestru musimy oznaczyć obraz adresem URL naszego lokalnego rejestru Docker Registry i nadać mu inną nazwę:

```
$ docker image tag alpine localhost:5000/localalpine
```

Teraz możemy załadować przygotowany obraz do lokalnego rejestru. W tym celu uruchomimy polecenie:

```
$ docker image push localhost:5000/localalpine
```

uruchomienie tego polecenia w wierszu poleceń spowoduje wyświetlenie następujących informacji:

```

russ in ~
⚡ docker image pull alpine
Using default tag: latest
latest: Pulling from library/alpine
2aec7e1714b: Pull complete
Digest: sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d81490948876e462c
Status: Downloaded newer image for alpine:latest
russ in ~
⚡ docker image tag alpine localhost:5000/localalpine
russ in ~
⚡ docker image push localhost:5000/localalpine
The push refers to a repository [localhost:5000/localalpine]
3f66f713c9f: Layer already exists
latest: digest: sha256:0b94d1d1b5eb130dd0253374552445b39470653fb1a1ec2d81490948876e462c size: 528
russ in ~
⚡ 

```

Wypróbuj działanie polecenia:

```
$ docker image ls
```

W oknie powinieneś znaleźć dwa obrazy o tym samym identyfikatorze (kolumna *IMAGE ID*):

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|----------------------------|--------|--------------|-------------|---------|
| alpine | latest | a41a7446062d | 3 weeks ago | 3.97 MB |
| localhost:5000/localalpine | latest | a41a7446062d | 3 weeks ago | 3.97 MB |
| registry | 2 | 9d0c4eabab4d | 5 weeks ago | 33.2 MB |

Przed pobraniem obrazu z lokalnego rejestru powinniśmy usunąć dwie lokalne kopie tego obrazu. Mamy dwa obrazy oznaczone takim samym identyfikatorem, a więc podczas usuwania powinniśmy podawać nazwy repozytoriów (jeżeli postąpimy inaczej, to Docker wyświetli komunikat informujący o błędzie):

```
$ docker image rm alpine localhost:5000/localalpine
```

Po usunięciu oryginalnego obrazu i obrazu ze zmienionym oznaczeniem możemy pobrać obraz z lokalnego rejestru. W tym celu musimy uruchomić polecenia:

```
$ docker image pull localhost:5000/localalpine
$ docker image ls
```

Jak widzisz, skopiowaliśmy obraz z rejestru Docker Registry działającego pod adresem `localhost:5000`:

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|----------------------------|--------|--------------|-------------|---------|
| localhost:5000/localalpine | latest | a41a7446062d | 3 weeks ago | 3.97 MB |
| registry | 2 | 9d0c4eabab4d | 5 weeks ago | 33.2 MB |

W celu zatrzymania działania rejestru i usunięcia go należy uruchomić następujące polecenia:

```
$ docker container stop registry  
$ docker container rm -v registry
```

Uruchamiając rejestr Docker Registry, powinno się przemyśleć wiele zagadnień. Jak zapewne się domyślasz, najważniejsze z nich dotyczą przechowywania. Główną funkcją rejestru jest przechowywanie i dystrybucja obrazów, a więc musisz zdecydować się na wybór miejsca przechowywania danych. Docker Registry obecnie obsługuje następujące opcje:

- **System plików:** obrazy mogą być przechowywane w wybranym miejscu systemu plików. Domyślną ścieżką jest */var/lib/registry*.
- **Azure:** chmura Microsoft Azure (<https://azure.microsoft.com/en-gb/services/storage/>).
- **GCS:** chmura Google Cloud Storage (<https://cloud.google.com/storage/>).
- **S3:** usługa Amazon Simple Storage Service — Amazon S3 (<https://aws.amazon.com/s3/>).
- **Swift:** OpenStack Swift (<https://wiki.openstack.org/wiki/Swift>).

Jak widzisz, Docker Registry umożliwia przechowywanie obrazów w lokalnym systemie plików, a także w wysoce dostępnych rozproszonych chmurach. Informacje na temat konfiguracji zaplecza przechowywania danych znajdziesz w oficjalnej dokumentacji znajdującej się pod adresem <https://docs.docker.com/registry/configuration/>.

Rejestr Docker Trusted Registry

Jednym z komponentów wchodzących w skład komercyjnego centrum danych Docker Datacenter jest zaufany rejestr Dockera — **Docker Trusted Registry (DTR)**. Rejestr ten należy postrzegać jako repozytorium Docker Hub, które może być uruchomione w lokalnej infrastrukturze. Docker Trusted Registry rozszerza możliwości oferowane przez darmowe narzędzia Docker Hub i Docker Registry o:

- Integrację z usługami uwierzytelniającymi, takimi jak Active Directory lub LDAP.
- Możliwość wdrożenia we własnej infrastrukturze (lub w chmurze) — za własną zaporą połączenia internetowego.
- Podpisywanie obrazów w celu potwierdzenia ich autentyczności.
- Wbudowane zabezpieczenie w postaci skanowania.
- Priorytetowy dostęp do pomocy Dockera.

Więcej informacji na temat Docker Trusted Registry znajdziesz na stronie <https://docs.docker.com/datacenter/dtr/2.2/guides/>.

Niezależne rejesty

Usługi przechowywania obrazów są świadczone nie tylko przez twórców Dockera. Firmy takie jak np. Red Hat udostępniają własne, niezależne repozytoria. Na stronie <https://access.redhat.com/containers/> znajdziesz kontenery zawierające produkty firmy Red Hat, a także kontenery obsługujące platformę OpenShift (<https://www.openshift.com>). Usługi takie jak Artifactory firmy JFrog (<https://jfrog.com/artifactory/>) oferują możliwość tworzenia prywatnych rejestrów Dockera.

Istnieją również rejesty w formie usług. Przyjrzymy się dwóm takim rozwiązaniom.

Quay

Quay jest komercyjną usługą oferowaną przez CoreOS. Usługa ta jest oferowana w wersji chmury (to właśnie tej wersji będziemy się przyglądać) i w wersji Enterprise, którą można uruchomić lokalnie — za zaporą połączenia internetowego.

The screenshot shows the Quay.io homepage with a dark blue background. At the top, there's a navigation bar with links for 'Tour', 'Tutorial', 'Pricing', 'Docs', and 'Blog'. On the right side of the bar are a search bar, a magnifying glass icon, and a 'Sign in' button. The main headline reads 'Quay [builds, analyzes, distributes] your container images'. Below it are two prominent buttons: 'TRY FOR FREE ON PREMISES' (in green) and 'TRY FOR FREE ON CLOUD' (in blue). Further down, there's a section stating 'Works with docker rkt' and 'Supports container image formats for both Docker and rkt'. A 'Contact Us' button is located at the bottom right of the page.

Na stronie <https://quay.io> możesz skorzystać bezpłatnie z wersji próbnej tej usługi przez okres 30 dni. W tym celu należy kliknąć przycisk *Sign In* (załóż konto) i następnie utworzyć nowe konto za pomocą konta serwisu GitHub. Po utworzeniu konta zostaniesz przywitany przez następującą stronę:

The screenshot shows the 'Repositories' page of the Quay.io application. At the top, there's a navigation bar with links for 'Applications', 'Repositories' (which is highlighted in red), 'Tutorial', 'Docs', and 'Blog'. On the right side of the bar are a search bar, a magnifying glass icon, a plus sign icon, a notification bell icon with a '1' notification, and a user profile icon for 'russmckendrick'. The main heading is 'Repositories'. Below it, there's a section titled 'Starred' with a message: 'You haven't starred any repositories yet.' It also says 'Stars allow you to easily access your favorite repositories.' Another message states 'This namespace doesn't have any viewable repositories.' At the bottom of this section, there's a note: 'Either no repositories exist yet or you may not have permission to view any. If you have permission, try [creating a new repository](#).' To the right of the main content area, there's a sidebar with 'Create New Repository' and 'Users and Organizations' sections. The 'Users and Organizations' section shows a profile for 'russmckendrick' and a 'Create New Organization' button. At the very bottom of the page, there's a footer with links for 'Contact' and 'All Systems Operational', along with a 'Contact Us' button.

W celu utworzenia nowego repozytorium, podobnie jak w przypadku automatyzacji budowy obrazów w serwisie Docker Hub, należy kliknąć przycisk *Create New Repository* (utwórz nowe repozytorium).

Spowoduje to wyświetlenie strony, na której zostaniesz poproszony o podanie nazwy Twojego nowego repozytorium. Teraz możesz skorzystać z kilku dostępnych opcji:

- *Empty repository* (utwórz puste repozytorium);
- *Initialize from a Dockerfile* (inicjalizuj na podstawie pliku Dockerfile);
- *Link to a GitHub repository push* (połącz z ładowaniem do repozytorium GitHub);
- *Link to a Bitbucket repository push* (połącz z ładowaniem do repozytorium Bitbucket);
- *Link to a GitLab repository push* (połącz z ładowaniem do repozytorium GitLab);
- *Link to a custom Git repository push* (połącz z ładowaniem do spersonalizowanego repozytorium Git).

Ponownie podepniemy się do konta w serwisie GitHub. W związku z tym wybieramy odpowiednią opcję i klikamy przycisk *Create Public Repository* (utwórz publiczne repozytorium). Spowoduje to przeniesienie na stronę, na której musimy upoważnić usługę Quay do dostępu do konta GitHub. Po udzieleniu pozwolenia musimy wskazać repozytorium GitHub, w którym znajduje się plik Dockerfile:

The screenshot shows the Quay.io interface for setting up a build trigger. At the top, it displays the user's organization: russmckendrick. Below this, the title 'Setup Build Trigger: 2c2b3352' is shown. The first section, 'Select organization', asks the user to choose an organization where the repository lives. It lists three organizations: russmckendrick (selected), a partially obscured organization, and another partially obscured organization. The second section, 'Select Repository', asks the user to choose a repository within the selected organization. It lists one repository: mastering-docker, which was updated 7 days ago with no description. A note indicates that a webhook will be added to detect new commits. At the bottom, there are navigation links for the build trigger setup process.

Teraz klikamy przycisk *Continue* (kontynuuj) i wybieramy ścieżkę pliku Dockerfile, określamy kontekst budowanego obrazu (wszystkie pliki umieszczone w obrazie kontenera) itd. Po zakończonej weryfikacji na ekranie pojawi się możliwość ręcznego uruchomienia procesu budowy obrazu. Po zbudowaniu obrazu wyświetlony zostanie następujący komunikat:

Teraz możemy pobrać wygenerowany obraz. Utworzony przez nas obraz można pobrać za pomocą polecenia:

```
$ docker image pull quay.io/russmckendrick/dockerfile-example
```

Oto komunikaty wyświetlane po uruchomieniu tego polecenia:

```
1. russ (bash)
russ in ~
⚡ docker image pull quay.io/russmckendrick/dockerfile-example
Using default tag: latest
latest: Pulling from russmckendrick/dockerfile-example
1f02c775bd9a: Pull complete
a3ed95caeb02: Pull complete
5e6712963917: Pull complete
34dc12b354cb: Pull complete
eda8ac5608ea: Pull complete
482a1e9a40e4: Pull complete
Digest: sha256:8a290eae41b2386a775dc8837265dbfa024e5aeaf587d11c27f73ab5e7982d16
Status: Downloaded newer image for quay.io/russmckendrick/dockerfile-example:latest
russ in ~
⚡ docker image ls
REPOSITORY          TAG      IMAGE ID   CREATED    SIZE
quay.io/russmckendrick/dockerfile-example  latest   b86d85361a4c  9 minutes  5.49 MB
```

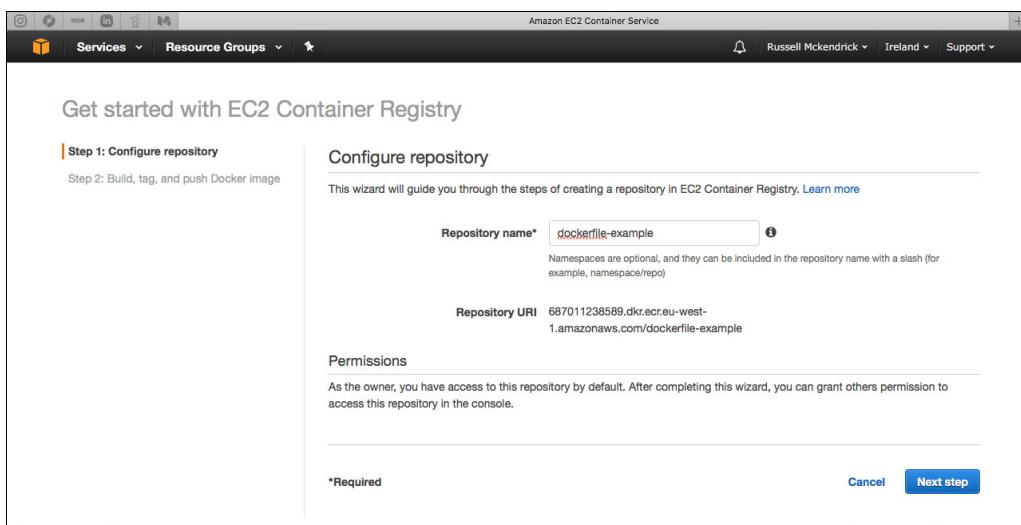
Jak widzisz, proces pobierania obrazu przebiega podobnie jak w przypadku pobierania obrazu z Docker Hub. Każda aktualizacja zawartości repozytorium GitHub spowoduje automatyczne uruchomienie budowy nowej wersji obrazu. O wystąpieniu ewentualnych problemów

użytkownik jest powiadamiany e-mailem. Dodatkową opcją jest możliwość pobierania spakowanych wersji automatycznie zbudowanych obrazów.Więcej informacji na temat usługi Quay znajdziesz na stronie <https://quay.io>.

Rejestr Amazon EC2 Container Registry

Amazon (twórcą usługi **Amazon EC2 Container Service (ECS)**) jest dostawcą kolejnej usługi, której chcemy przyjrzeć się bliżej. Usługa rejestru **Amazon EC2 Container Registry (ECR)** działa nieco inaczej od usług Docker Hub i Quay.

Usługę tę można znaleźć po kliknięciu przycisku *Services* (usługi) znajdującego się w górnym lewym rogu strony *Amazon Web Services* i wpisaniu hasła *ECS*. Po załadowaniu kolejnej strony kliknij przycisk *Repositories* (repozytoria) i nadaj nazwę swojemu repozytorium:



Może Ci się wydawać, że po kliknięciu przycisku *Next step* (dalej) strona poprosi Cię o wprowadzenie szczegółów repozytorium GitHub, Bitbucket lub Amazon S3 zawierającego plik Dockerfile, ale czeka Cię coś zupełnie innego. Na kolejnej stronie będą znajdowały się instrukcje dotyczące budowy, nadawania nazw i etykiet, a także ładowania obrazów. W moim przypadku instrukcja ta miała formę:

```
$ aws ecr get-login --no-include-email --region eu-west-1
```

Instrukcja ta generuje polecenie pozwalające na zalogowanie się. W moim przypadku polecenie to wyglądało następująco (skróciłem hasło, które miało ponad 1500 znaków):

```
docker login -u AWS -p
eyJwYX1sb2FkIjoiUEg5dmZ0RV1Kb2RRSjRBaEdtOUpxQWkydURVaGFEZ3JYRXIrQ11VQi8yejN3VXpxWjg
↳3YUdCeng0VjBiR0F1L2ZNMoTHQWSJ9
https://687011138589.dkr.ecr.eu-west-1.amazonaws.com
```

Po zalogowaniu możesz budować obrazy i nadawać im etykiety (musisz korzystać z określonych przez Amazon instrukcji). Ja korzystałem z następujących przykładowych polecień:

```
$ docker image build -t dockerfile-example .
$ docker image tag dockerfile-example:latest 687011238589.dkr.ecr.euwest-1.
  ↪amazonaws.com/dockerfile-example:latest
$ docker image push 687011238589.dkr.ecr.euwest-1.amazonaws.com/dockerfile-
  ↪example:latest
```

Załadowany obraz jest wyświetlany w konsoli AWS:

| Image tags | Digest | Size (MIB) | Pushed at |
|------------|--|------------|---------------------------|
| latest | sha256:9f2c748054a07807dd252263505464... | 2.52 | 2017-06-18 15:34:58 +0100 |

Po zmodyfikowaniu zabezpieczeń mogę udostępnić ten obraz do pobrania każdej osobie, która uruchomi polecenie:

```
$ docker image pull 687011238589.dkr.ecr.euwest-1.amazonaws.com/dockerfile-example
```

Jak widzisz, to prosta usługa umożliwiająca ładowanie i pobieranie obrazów. Nie ma tutaj standardowej opcji automatycznego budowania obrazów, ale istnieje możliwość utworzenia własnego potoku składającego się z polecień wymaganych do zalogowania się, a także zbudowania i załadowania obrazu.

Usługom obsługującym kontenery dostarczanym przez firmę Amazon przyjrzymy się bliżej w kolejnych rozdziałach. Więcej informacji na temat rejestru Amazon EC2 Container Registry znajdziesz na stronie <https://aws.amazon.com/ecr/>.

Microbadger

Microbadger to świetne narzędzie przydatne podczas udostępniania kontenerów i obrazów. Bierze ono pod uwagę wszystko to, co dzieje się w każdej warstwie obrazu Dockera, i wyświetla informacje o rozmiarze każdego elementu obrazu.

The screenshot shows the MicroBadger website. At the top, there's a navigation bar with links for Labels, Private registries, Support, About, and a user profile for Russ McKendrick. A red "BETA" badge is visible. The main heading is "Manage your Docker container images". Below it is a search bar with placeholder text "e.g. centos or puppet/puppetserver" and a "View image" button. Three main sections are displayed: "Inspect" (with a magnifying glass icon), "Stay up-to-date" (with a clock icon), and "Badges" (with a flag icon). Each section has a brief description and a call-to-action. A small blue box at the bottom right indicates "14,605 deployed".

Na powyższym rysunku pokazano stronę narzędzia Microbadger znajdującą się pod adresem <https://microbadger.com>.

Strona ta umożliwia wyszukiwanie obrazów udostępnianych w serwisie Docker Hub w celu przeanalizowania ich zawartości. Ponadto możesz załadować własny zestaw obrazów do analizy lub przyjrzeć się zestawom obrazów opracowanych przez innych użytkowników Dockera.

W tym przykładzie poszukamy naszego obrazu aplikacji *cluster* i wybierzemy jego najnowszą wersję (etykieta *latest*). Jak widzisz na poniższym rysunku, serwis Docker Hub jest automatycznie przeszukiwany podczas wpisywania nazwy obrazu:

This screenshot shows the MicroBadger interface after searching for the 'cluster' image. The search bar now contains 'russmckendrick/cluster'. The 'Inspect' section shows the result for 'russmckendrick/cluster' with a magnifying glass icon. The 'Stay up-to-date' section is still present with its clock icon. The 'Badges' section remains the same with its flag icon. The "14,605 deployed" badge is also visible.

Domyślnie ładowany jest obraz oznaczony etykietą *latest* (najnowszy), ale możesz wybrać również inną wersję obrazu — wystarczy skorzystać z rozwijanego menu *Versions* (wersje). Przydaje się to podczas sprawdzania objętości obrazów oznaczonych etykietą finalnej wersji testowej.

Microbadger przedstawia informacje na temat tego, z ilu warstw składa się obraz:

Metadata from image russmckendrick/cluster

Last inspected 4 hours ago.

[Versions ▾](#)

| | |
|---|---------------------------------|
| Tags | latest |
| Created | March 03, 2017 at 08:39 PM |
| ID | 98369be7ee59 |
| Maintainer | Russ McKendrick <russ@[hidden]> |
| Download Size | 15.5 MB |
| Labels | No labels |
| Layers | 14 |
| 3.2 MB russmckendrick/base latest <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> ADD file:730030a984f5f0c5dc9b15ab61da161082b5c0f6e112a9c921b423 21140c3927 in / </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> MAINTAINER Russ McKendrick <russ@[hidden]> </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> RUN apk update && apk upgrade && apk add ca-certificates bash && rm -rf /var/cache/apk/* </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> MAINTAINER Russ McKendrick <russ@[hidden]> </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> RUN apk add --update supervisor nginx && rm -rf /var/cache/apk/* && mkdir -p /var/www/html </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> COPY multi:3bdc7274da3d28b1a2c8bd5fbeb884d5a7d00e75c27968fa2e91997f150bae35 in /var/www/html/ </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> COPY file:ac42ad54f2bb25ce997565f5423fbcc7c11d03fd6d62a63d7863b88892cbbc62 in /script/ </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> COPY file:b77a553c5566a605e475fe61f9c9f224c668c732b3b95dce6a444a47bea166bb in /etc/nginx/conf.d/ </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> COPY file:90d3709a1dcba3b7d65e1be3bd884f348742c636afc248e3c343da90f03d8bfa4 in /etc/nginx/nginx.conf </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> COPY file:4f254ac566b6c11cb2cc197c96e510955f030bbf28c9502b826c4230a3db372b in /etc/supervisord.conf </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> RUN chown -R nginx:nginx /var/www/html </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> EXPOSE 80/tcp </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> ENTRYPOINT ["supervisord"] </div> <div style="background-color: #337ab7; color: white; padding: 5px; margin-top: -10px;"> CMD ["-c" "/etc/supervisord.conf"] </div> | |

Ponadto w danych można znaleźć informacje o objętości każdej warstwy i poleceniach pliku Dockerfile wykonywanych podczas budowy obrazu. Wszystko to jest bardzo przydatne podczas zmniejszania objętości obrazu — możesz wykryć warstwy, które przyczyniają się do nadmiernego rozrostu obrazu.

Microbadger umożliwia również przekazywanie podstawowych informacji na temat obrazów do repozytorium Git lub Docker Hub. Oto wygląd strony zawierającej takie informacje na temat obrazu cluster umieszczonego w serwisie Docker Hub (<https://hub.docker.com/r/russmckendrick/cluster/>):

The screenshot shows the Docker Hub interface for the repository 'russmckendrick/cluster'. At the top, there's a search bar and navigation links for Dashboard, Explore, Organizations, Create, and the user profile 'russmckendrick'. Below that, it says 'PUBLIC | AUTOMATED BUILD' and displays the repository name 'russmckendrick/cluster' with a star icon. It indicates the last push was '2 months ago'. A horizontal menu bar includes 'Repo Info' (which is selected), Tags, Dockerfile, Build Details, Build Settings, Collaborators, Webhooks, and Settings. The main content area has two sections: 'Short Description' containing 'A container to demonstrate clusters' and 'Full Description' containing 'Cluster Example' (with a size of 15.5MB and 14 layers), a note about running NGINX, and a command line snippet: 'docker run -d -p 80:80 russmckendrick/cluster'. To the right, there's a sidebar with 'Docker Pull Command' (containing 'docker pull russmckendrick/cluster'), 'Owner' (with a profile picture and 'russmckendrick'), and 'Source Repository' (with a link to 'russmckendrick/docker').

Microbadger wyświetla informacje o całkowitym rozmiarze obrazu i liczbie tworzących go warstw. Usługa Microbadger jest wciąż w fazie wersji beta. Stale dodawane są do niej nowe funkcje. Warto przyglądać się jej dalszemu rozwojowi.

Podsumowanie

Opisaliśmy kilka sposobów ręcznego i automatycznego budowania obrazów kontenerów. Korzystaliśmy z usług takich jak Docker Hub, Quay i Amazon ECR. Przedstawiliśmy również kilka rejestrów, które stanowią alternatywę serwisu Docker Hub (były to Docker Store i katalog kontenerów firmy Red Hat).

Ponadto opisaliśmy zagadnienia związane z wdrażaniem własnego lokalnego rejestru Docker Registry i przedstawiliśmy sytuacje, w których takie rozwiązanie jest przydatne. Na koniec przyjrzaliśmy się usłudze Microbadger, która wyświetla informacje o obrazach kontenerów umieszczonych w zdalnych repozytoriach.

W kolejnym rozdziale zajmiemy się zarządzaniem kontenerami z poziomu wiersza poleceń.

Zarządzanie kontenerami

Dotychczas skupialiśmy się na zagadnieniach związanych z budową, przechowywaniem i dystrybucją obrazów Dockera. Teraz zajmiemy się uruchamianiem kontenerów i zarządzaniem nimi z poziomu wiersza poleceń Dockera.

Przyjrzymy się poleceniom użytym w rozdziale 1., a także zgłębimy inne dostępne polecenia. Po przedstawieniu poleceń służących do obsługi kontenerów zajmiemy się zagadnieniami związanymi z sieciami i wolumenami Dockera.

W tym rozdziale zajmiemy się następującymi tematami:

- Polecenia obsługujące kontenery Dockera:
 - podstawy,
 - praca z kontenerami,
 - dzienniki i informacje o procesach,
 - ograniczanie zasobów,
 - stany kontenerów i polecenia pomocnicze,
 - usuwanie kontenerów.
- Obsługa sieci i wolumenów.

Polecenia służące do obsługi kontenerów Dockera

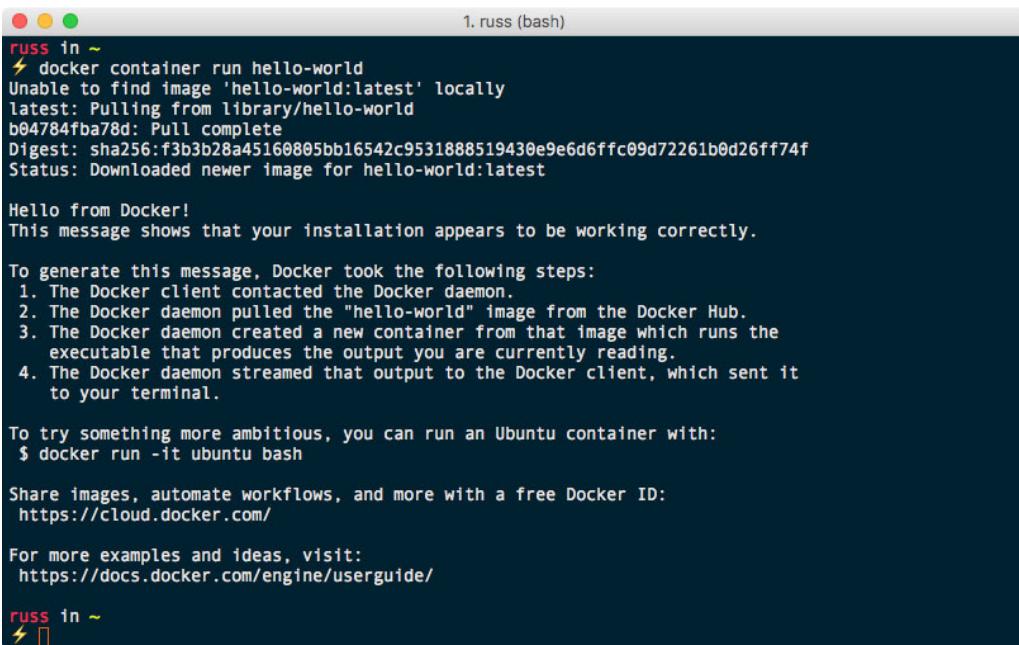
Zanim zajmiemy się bardziej złożonymi poleceniami Dockera, chcielibyśmy wrócić do polecień używanych w poprzednich rozdziałach i opisać je dokładniej.

Podstawy

W rozdziale 1. „Docker — wprowadzenie” uruchomiliśmy najprostszy kontener Dockera (kontener `hello-world`):

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

Oto zawartość okna Terminala wygenerowana w wyniku uruchomienia tego polecenia:



```
russ in ~
⚡ docker container run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b04784fba78d: Pull complete
Digest: sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26ff74f
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.
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://cloud.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/
```

Polecenie to pobiera obraz o rozmiarze 1,84 kB z serwisu Docker Hub (więcej informacji na temat tego obrazu znajdziesz na stronie <https://store.docker.com/images/hello-world/>), a następnie zgodnie z zawartością pliku Dockerfile uruchamia plik wykonywalny o nazwie `hello`:

```
FROM scratch
COPY hello /
CMD ["/hello"]
```

Plik ten wyświetla komunikat o treści *Hello from Docker!* (witaj w Dockerze) w Terminalu, a następnie zamyka swój proces. Kontener jest zatrzymywany wraz z wyjściem z tego procesu. Można to sprawdzić za pomocą następującego polecenia:

```
$ docker container ls -a
```

Oto informacje wyświetcone przez to polecenie:

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|-------------|----------|------------|---------------------------|-------|--------------------|
| 9243b527c8c2 | hello-world | "/hello" | 2 days ago | Exited (0) 11 minutes ago | | infallible_davinci |

Na rysunku widać, że najpierw uruchomiłem polecenie `docker container ls` bez flagi `-a` (skrótu flagi `-all`), ale bez tej flagi polecenie nie wyświetla zatrzymanych kontenerów.

Nie musielibyśmy zmieniać nazwy tego kontenera, ponieważ używaliśmy go bardzo krótko. Docker automatycznie przypisuje nazwy kontenerom. W moim przypadku kontenerowi nadano nazwę `infallible_davinci`.

Korzystając dłużej z Dockera, zauważysz, że mechanizm automatycznego generowania nazw generuje naprawdę ciekawe frazy. Jeżeli Cię to interesuje, to kod generujący nazwy znajdziesz na stronie <https://github.com/moby/moby/blob/master/pkg/namesgenerator/names-generator.go>. Zauważ, że na jego końcu znajduje się następująca instrukcja warunkowa `if`:

```
if name == "boring_wozniak" /*Steve Wozniak is not boring */ {
    goto begin
}
```

W związku z tym nigdy nie będziesz miał kontenera o nazwie `boring_wozniak` (nudny Wozniak). To dobrze, Steve Wozniak wcale nie jest nudną osobą.

Steve Wozniak jest wynalazcą, inżynierem elektronikiem, programistą i przedsiębiorcą, który wraz ze Stevem Jobsem założył spółkę Apple Inc. Jest uważany za pioniera rewolucji komputerów osobistych, do której doszło w latach 70. i 80. Z pewnością nie jest to osoba, którą można określić mianem nudnej!

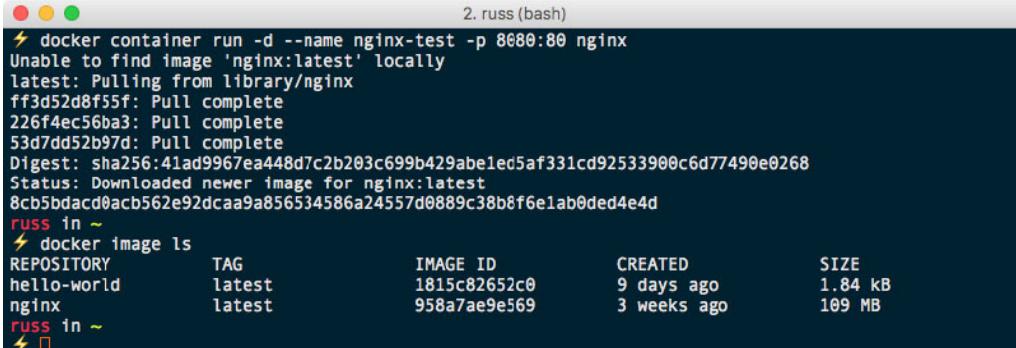
Zatrzymany kontener możemy usunąć za pomocą polecenia:

```
$ docker container rm infallible_davinci
```

Pod koniec rozdziału 1. „Docker — wprowadzenie” uruchomiliśmy kontener na podstawie oficjalnego obrazu NGINX (więcej informacji na temat tego obrazu znajdziesz na stronie <https://store.docker.com/images/nginx/>). Korzystaliśmy wtedy z polecenia:

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

Przypominamy, że polecenie to pobierało obraz, uruchamiało go, mapowało port 8080 hosta do portu 80 kontenera i wywoływało program nginx-test:

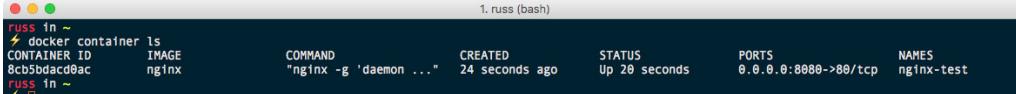


```
2. russ (bash)
⚡ docker container run -d --name nginx-test -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
ff3d52d8f55f: Pull complete
226f4ec56ba3: Pull complete
53d7dd52b97d: Pull complete
Digest: sha256:41ad9967ea448d7c2b203c699b429abed5af331cd92533900c6d77490e0268
Status: Downloaded newer image for nginx:latest
8cb5bdacd0acb562e92dcaa9a856534586a24557d0889c38b8f6e1ab0ded4e4d
russ in ~
⚡ docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
hello-world         latest   1815c82652c0  9 days ago    1.84 kB
nginx               latest   958a7ae9e569  3 weeks ago   109 MB
russ in ~
⚡
```

Jak widzisz, polecenie docker image ls informuje o tym, że mamy dwa pobrane i uruchomione kontenery.

```
$ docker container ls
```

Polecenie to informuje nas o tym, że mamy uruchomiony kontener. Z informacji widocznych na poniższym zrzucie wynika, że skorzystałem z niego 20 sekund po uruchomieniu kontenera:



```
1. russ (bash)
russ in ~
⚡ docker container ls
CONTAINER ID        IMAGE           COMMAND           CREATED          STATUS          PORTS          NAMES
8cb5bdacd0ac      nginx           "nginx -g 'daemon ..."   24 seconds ago   Up 20 seconds   0.0.0.0:8080->80/tcp   nginx-test
russ in ~
⚡
```

Do polecenia docker container run dodaliśmy trzy flagi. Jedną z nich była flaga -d. Jest to skrót od -detach (odłącz). Gdybyśmy nie dodali tej flagi, to kontener zostałby uruchomiony na pierwszym planie, co oznaczałoby zatrzymanie Terminala aż do momentu przekazania procesowi specjalnego polecenia poprzez wcisnięcie kombinacji klawiszy *Ctrl+C*.

Możemy to sprawdzić, uruchamiając drugi kontener nginx za pomocą następującego polecenia:

```
$ docker container run --name nginx-foreground -p 9090:80 nginx
```

Po uruchomieniu kontenera otwórz przeglądarkę i wpisz w niej adres <http://localhost:9090/>. Informacja o odwiedzeniu strony zostanie wyświetlona w oknie Terminala. Odświeżanie strony będzie powodować pojawianie się kolejnych komunikatów aż do momentu wcisnięcia kombinacji klawiszy *Ctrl+C*, dzięki której wróciszesz do Terminala.

```

⚡ docker container run --name nginx-foreground -p 9090:80 nginx
172.17.0.1 - [24/Jun/2017:10:34:09 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4"
172.17.0.1 - [24/Jun/2017:10:34:10 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4"
172.17.0.1 - [24/Jun/2017:10:34:10 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4"
172.17.0.1 - [24/Jun/2017:10:34:11 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4"
172.17.0.1 - [24/Jun/2017:10:34:14 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4"
172.17.0.1 - [24/Jun/2017:10:34:14 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4"
172.17.0.1 - [24/Jun/2017:10:34:15 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4"
172.17.0.1 - [24/Jun/2017:10:34:15 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4"
172.17.0.1 - [24/Jun/2017:10:34:15 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4"
^Cruss in ~
⚡

```

Po uruchomieniu polecenia docker container ls -a zobaczyś listę zawierającą dwa kontenery (z jednego z nich przed chwilą wyszedłeś):

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|-------|------------------------|----------------|--------------------------|----------------------|------------------|
| 5a254827f01d | nginx | "nginx -g 'daemon ...' | 3 minutes ago | Exited (0) 3 minutes ago | 0.0.0.0:8080->80/tcp | nginx-foreground |
| 8cb5bdacd9ac | nginx | "nginx -g 'daemon ...' | 21 minutes ago | Up 21 minutes | | nginx-test |

```

⚡ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
5a254827f01d        nginx              "nginx -g 'daemon ..."
8cb5bdacd9ac      nginx              "nginx -g 'daemon ...
russ in ~
⚡

```

Co się stało? Po usunięciu flagi -d Docker połączył nas bezpośrednio z procesem nginx uruchomionym w kontenerze, a więc widzieliśmy dane wejścia, wyjścia i błędów tego procesu (procedury stdin, stdout i stderr). Wcisnąć kombinację klawiszy **Ctrl+C**, tak naprawdę wysłaliśmy polecenie zakończenia procesu nginx. Proces ten utrzymywał działanie kontenera, a więc po jego zakończeniu Docker natychmiast wyszedł z kontenera.

Procedura standardowego wejścia (stdin) umożliwia procesowi uzyskanie dostępu do danych wprowadzanych przez użytkownika.

Procedura standardowego wyjścia (stdout) służy do uzyskiwania dostępu do standardowych informacji generowanych przez proces.

Procedura standardowego błędu (stderr) służy do przechwytywania komunikatów informujących o błędach.

Uruchamiając drugi kontener, nadaliśmy mu inną nazwę (**nginx-foreground**) — zrobiliśmy to za pomocą flagi **--name**.

Flagę tę zastosowaliśmy, ponieważ Docker nie pozwala na uruchomienie dwóch kontenerów o tej samej nazwie, ale pozwala na interakcję z kontenerami przy użyciu ich nazw (`NAME`) i identyfikatorów (`CONTAINER ID`). To właśnie dlatego w Dockerze zaimplementowano funkcję generującą losowe nazwy kontenerów, których nie chcesz nazywać samodzielnie. Dodatkowo funkcja ta dba o to, aby Steve Wozniak nigdy nie został nazwany osobą nudną.

Na koniec zwróćmy uwagę na to, że uruchamiając kontener `nginx-foreground`, poinformowaliśmy Dockera o tym, że port 9090 hosta powinien być połączony z portem 80 kontenera. Wynika to z braku możliwości przypisania kilku procesów do jednego portu hosta. Gdybyśmy tego nie zrobili, to próba uruchomienia drugiego kontenera korzystającego z tego samego portu skończyłaby się wyświetleniem komunikatu błędu:

```
docker: Error response from daemon: driver failed programming external connectivity  
↳ on endpoint nginx-foreground  
(c338635413d3aeae574e79a8eb89ede6a80a3ea4702de4800da89d0438a90be5): Bind for  
↳ 0.0.0.0:8080 failed: port is already allocated.
```

Kontener jest uruchamiany na pierwszym planie, a więc proces `nginx` nie mógłby zostać uruchomiony i wyświetliłby błąd:

```
ERRO[0003] error getting events from daemon: net/http: request cancelled
```

Mapujemy port 80 obu kontenerów, dlaczego nie powoduje to błędów? Zgodnie z tym, co wyjaśniliśmy w rozdziale 1. „Docker — wprowadzenie”, kontenery dysponują odizolowanymi zasobami, a więc możemy uruchomić dowolną liczbę kontenerów i dokonać przekierowania portów 80 wszystkich kontenerów. Problematyczne jest jedynie przekierowanie odsłoniętych portów kontenera z hostu Dockera. Pozostawmy kontener `nginx` aktywny podczas lektury jeszcze jednej sekcji.

Komunikowanie się z kontenerami

Dotychczas w kontenerach uruchamialiśmy tylko pojedyncze procesy. Docker pozwala na uruchamianie wielu procesów w kontenerze, a także komunikowanie się z nimi.

Attach

Interakcję z aktywnym kontenerem można nawiązać za pomocą polecenia `attach` podłączającego się do aktywnego procesu. Nie zatrzymaliśmy kontenera `nginx-test`, a więc podłączmy się do niego za pomocą polecenia:

```
$ docker container attach nginx-test
```

Po otwarciu przeglądarki i wejściu na stronę `http://localhost:8080/` zobaczymy informacje o próbach dostępu (tak samo jak w przypadku kontenera `nginx-foreground`). Wciśnięcie kombinacji klawiszy `Ctrl+C` przerwie działanie tego procesu i przywróci standardowe funkcjonowanie okna Terminala, ale spowodowałoby to ponownie przerwanie pracy kontenera:

```
russ in ~
⚡ docker container attach nginx-test
172.17.0.1 - - [24/Jun/2017:13:03:53 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
172.17.0.1 - - [24/Jun/2017:13:03:54 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
172.17.0.1 - - [24/Jun/2017:13:03:54 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
^Cruss in ~
⚡
```

Kontener możemy uruchomić ponownie za pomocą polecenia:

```
$ docker container start nginx-test
```

Kontener zostanie ponownie uruchomiony w stanie odłączonym (będzie pracował w tle). Po wejściu na stronę <http://localhost:8080> ponownie zobaczysz stronę powitalną kontenera nginx.

Podłączmy się ponownie do procesu. Tym razem przekażmy dodatkową opcję:

```
$ docker container attach --sig-proxy=false nginx-test
```

Kilkukrotne odświeżenie strony i wcisnięcie kombinacji klawiszy *Ctrl+C* odłączy nas od procesu, ale tym razem proces nie zostanie przerwany. Po prostu wróćmy do Terminala, a kontener będzie działał dalej w tle. Można to sprawdzić, uruchamiając polecenie `docker container ls`:

```
russ in ~
⚡ docker container attach --sig-proxy=false nginx-test
172.17.0.1 - - [24/Jun/2017:13:11:50 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
172.17.0.1 - - [24/Jun/2017:13:11:51 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
172.17.0.1 - - [24/Jun/2017:13:11:51 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
^C
russ in ~
⚡ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
8cb5bdacd0ac      nginx              "nginx -g 'daemon ..."   2 hours ago       Up 2 minutes
          0.0.0.0:8080->80/tcp
russ in ~
⚡
```

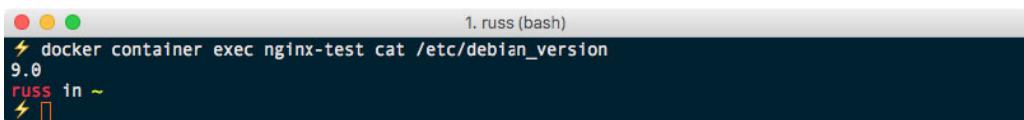
exec

Polecenie `attach` przydaje się w przypadku potrzeby podłączenia się do procesu uruchomionego w kontenerze, ale co, gdybyśmy chcieli zrobić coś w sposób bardziej interaktywny? Możemy skorzystać z polecenia `exec` uruchamiającego w kontenerze kolejny proces, z którym możesz również nawiązać interakcję.

Poniższe polecenie pozwala na podgląd zawartości pliku `/etc/debian_version`:

```
$ docker container exec nginx-test cat /etc/debian_version
```

W kontenerze uruchomiony zostaje drugi proces — polecenie cat, które przekazuje zawartość pliku `/etc/debian_version` do wyjścia `stdout`. Drugi proces zostanie od razu zakończony, a kontener będzie działał tak samo jak przed wykonaniem polecenia exec:



```
1. russ (bash)
⚡ docker container exec nginx-test cat /etc/debian_version
9.0
russ in ~
```

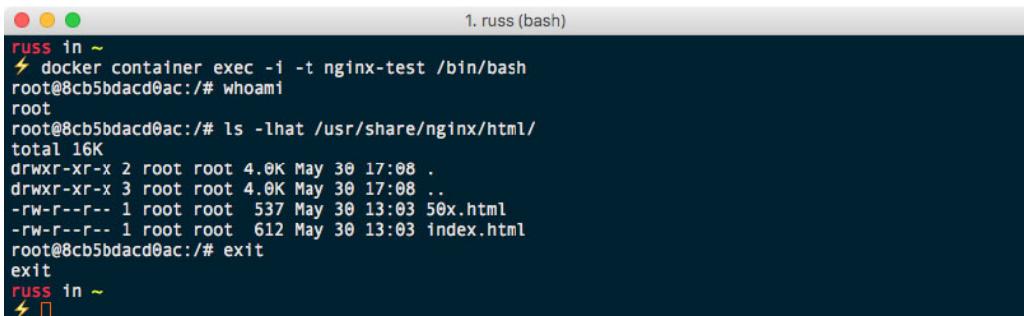
Możemy pójść o krok dalej i uruchomić następujące polecenie:

```
$ docker container exec -i -t nginx-test /bin/bash
```

Tym razem uruchamiamy proces basha i korzystamy z flag `-i` i `-t` w celu utrzymania dostępu do konsoli kontenera. Flaga `-i` jest skrótem flagi `--interactive`, która utrzymuje otwarty strumień danych `stdin`, co pozwala nam na przesyłanie poleceń do procesu. Flaga `-t` jest skrótem flagi `--tty`, która alokuje interfejs pseudo-TTY do sesji.

Pierwsze terminale użytkownika podłączane do komputerów określano mianem dalekopisów (ang. *teletypewriter*). Urządzenia te nie są obecnie używane, ale akronim ich nazwy (TTY) jest w dalszym ciągu używany w odniesieniu do konsol tekstowych współczesnych komputerów.

W związku z tym możesz komunikować się z kontenerem tak, jakbyś korzystał ze zdalnej sesji Terminala, np. poprzez protokół SSH:



```
1. russ (bash)
russ in ~
⚡ docker container exec -i -t nginx-test /bin/bash
root@8cb5bdacd0ac:/# whoami
root
root@8cb5bdacd0ac:/# ls -lhat /usr/share/nginx/html/
total 16K
drwxr-xr-x 2 root root 4.0K May 30 17:08 .
drwxr-xr-x 3 root root 4.0K May 30 17:08 ..
-rw-r--r-- 1 root root 537 May 30 13:03 50x.html
-rw-r--r-- 1 root root 612 May 30 13:03 index.html
root@8cb5bdacd0ac:/# exit
exit
russ in ~
```

Komunikowanie się z kontenerem tak, jakby był on maszyną wirtualną, jest bardzo praktyczne, ale odradzam Ci wprowadzanie zmian w aktywnych kontenerach za pomocą interfejsu pseudo-TTY. Istnieje zagrożenie, że wprowadzone w ten sposób zmiany nie zostaną zachowane i stracisz je w wyniku usunięcia kontenera. Temat ten zglebimy w rozdziale 12. „Przepływy zadań w platformie Docker”.

Dzienniki i informacje o procesach

Dotychczas w celu uzyskania danych podłączaliśmy się bezpośrednio do kontenera lub uruchomionego w nim procesu. Docker obsługuje również kilka poleceń umożliwiających podgląd informacji o kontenerach bez używania poleceń attach lub exec.

logs

Polecenie logs pozwala na uzyskanie dostępu do strumienia danych wyjściowych stdout kontenerów, które są rejestrowane w tle przez Dockera. Oto przykładowe polecenie umożliwiające podgląd ostatnich wpisów dziennika strumienia danych wyjściowych stdout kontenera nginx-test:

```
$ docker container logs --tail 5 nginx-test
```

Oto dane zwrócone przez to polecenie:

```
russ in ~
$ docker container logs --tail 5 nginx-test
172.17.0.1 - - [24/Jun/2017:13:03:54 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
172.17.0.1 - - [24/Jun/2017:13:11:50 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
172.17.0.1 - - [24/Jun/2017:13:11:51 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
172.17.0.1 - - [24/Jun/2017:13:11:51 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
172.17.0.1 - - [24/Jun/2017:13:31:27 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "
russ in ~
```

W celu podglądzania dziennika danych wyjściowych w czasie rzeczywistym należy uruchomić polecenie:

```
$ docker container logs -f nginx-test
```

Flaga -f to skrót flagi --follow (śledź). Oto instrukcja, w wyniku której możliwe jest przesyłanie wpisów dziennika dokonanych dzisiaj od godziny 15:00:

```
$ docker container logs --since 2017-06-24T15:00 nginx-test
```

Oto dane wyświetlane po uruchomieniu tego polecenia:

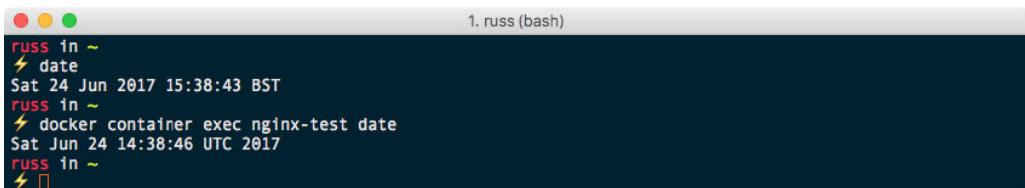
```
russ in ~
$ docker container logs --since 2017-06-24T15:00 nginx-test
172.17.0.1 - - [24/Jun/2017:14:34:04 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "-"
172.17.0.1 - - [24/Jun/2017:14:34:04 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Safari/603.2.4" "
russ in ~
```

Jeden z wyświetlonych wpisów dziennika dostępu charakteryzuje się znacznikiem czasu 14:34, a więc wpisu tego dokonano przed 15:00. Dlaczego w wyświetlonych danych uwzględniono ten wpis?

Polecenie `logs` pokazuje znaczniki czasu wyjścia `stdout` zapisane przez Dockera (nie jest to czas wewnętrzny kontenera). Możemy to sprawdzić, uruchamiając następujące polecenia:

```
$ date  
$ docker container exec nginx-test date
```

Wygenerują one następujące informacje:



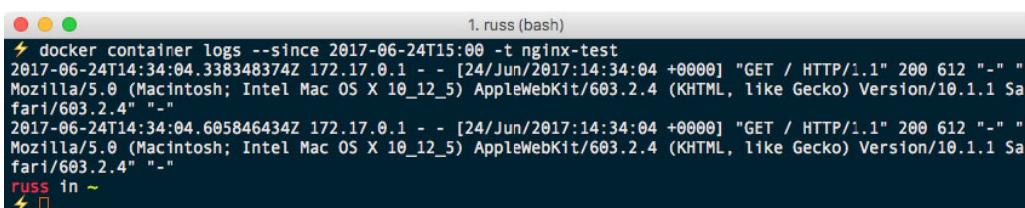
```
russ in ~  
⚡ date  
Sat 24 Jun 2017 15:38:43 BST  
russ in ~  
⚡ docker container exec nginx-test date  
Sat Jun 24 14:38:46 UTC 2017  
russ in ~  
⚡
```

Różnica czasu pomiędzy moim hostem a kontenerem wynika z tego, że kontener przyjął czas uniwersalny (UTC), a mój komputer czas letni właściwy dla Wielkiej Brytanii (BST).

Problem ten można rozwiązać, dodając do polecenia `logs` flagę `-t`:

```
$ docker container logs --since 2017-06-24T15:00 -t nginx-test
```

Flaga `-t` jest skrótem flagi `--timestamp` (znacznik czasu), która dodaje czas odebrania danych wyjściowych przez Dockera:



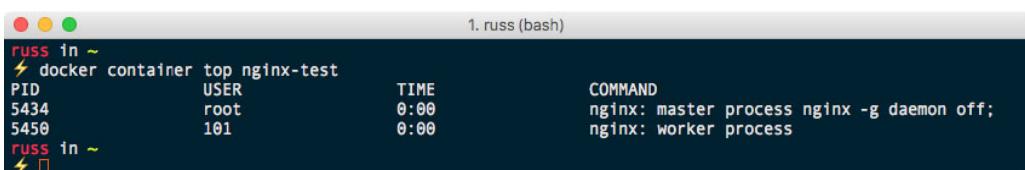
```
⚡ docker container logs --since 2017-06-24T15:00 -t nginx-test  
2017-06-24T14:34:04.338348374Z 172.17.0.1 - - [24/Jun/2017:14:34:04 +0000] "GET / HTTP/1.1" 200 612 "-" "  
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Sa  
fari/603.2.4" "-"  
2017-06-24T14:34:04.605846434Z 172.17.0.1 - - [24/Jun/2017:14:34:04 +0000] "GET / HTTP/1.1" 200 612 "-" "  
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/603.2.4 (KHTML, like Gecko) Version/10.1.1 Sa  
fari/603.2.4" "-"  
russ in ~  
⚡
```

top

Polecenie `top` wyświetla listę procesów uruchomionych w wybranym kontenerze:

```
$ docker container top nginx-test
```

Oto dane wyświetcone w wyniku uruchomienia tego polecenia:



```
russ in ~  
⚡ docker container top nginx-test  
PID USER TIME COMMAND  
5434 root 0:00 nginx: master process nginx -g daemon off;  
5450 101 0:00 nginx: worker process  
russ in ~  
⚡
```

Z wyświetlonych informacji wynika, że w kontenerze uruchomione są zgodnie z oczekiwaniemi dwa procesy serwera nginx.

stats

Polecenie stats służy do wyświetlania w czasie rzeczywistym informacji o wybranym kontenerze. W przypadku niepodania nazwy kontenera lub jego identyfikatora polecenie to wyświetli informacje o wszystkich aktywnych kontenerach:

```
$ docker container stats nginx-test
```

Spójrz na poniższy zrzut. Jak widzisz, zwracane są informacje o obciążeniu procesora (CPU), pamięci (RAM), sieci (NETWORK) i operacjach wejścia-wyjścia (I/O):

| 1. russ (docker) | | | | | | |
|------------------|-------|-----------------------|-------|-----------------|-----------|------|
| CONTAINER | CPU % | MEM USAGE / LIMIT | MEM % | NET I/O | BLOCK I/O | PIDS |
| nginx-test | 0.00% | 1.844 MiB / 1.952 GiB | 0.09% | 1.38 kB / 928 B | 0 B / 0 B | 2 |

Polecenie to obsługuje również flagę `-a` (`--all`) służącą do wyświetlania informacji o wszystkich kontenerach (również tych nieaktywnych):

| 1. russ (docker) | | | | | | |
|------------------|-------|-----------------------|-------|----------------|-----------|------|
| CONTAINER | CPU % | MEM USAGE / LIMIT | MEM % | NET I/O | BLOCK I/O | PIDS |
| e6ac3ce1418d | 0.00% | 1.844 MiB / 1.952 GiB | 0.09% | 1.7 kB / 998 B | 0 B / 0 B | 2 |
| 03eaa561e8cf | 0.00% | 0 B / 0 B | 0.00% | 0 B / 0 B | 0 B / 0 B | 0 |

Jak widzisz, nieaktywny kontener nie obciąża żadnych zasobów, a więc dodanie tej flagi pozwoli tylko na zorientowanie się w tym, ile kontenerów zostało uruchomionych i ile z nich korzysta z zasobów komputera.

Warto pamiętać o tym, że polecenie stats wyświetla informacje w czasie rzeczywistym. Docker nie zapisuje informacji o obciążeniu zasobów sprzętowych, a więc nie można ich przeglądać tak jak dziennik za pomocą polecenia logs.

Ograniczenia zasobów

Ostatnie polecenie, które uruchamialiśmy, pokazywało informacje o obciążeniu zasobów przez kontenery. Domyślnie uruchomiony kontener może korzystać ze wszystkich dostępnych zasobów hosta. Możemy ograniczyć zasoby, z których korzysta kontener. Zróbjmy to na przykładzie kontenera nginx-test.

Zwykłe ograniczenia dotyczące możliwości korzystania z zasobów sprzętowych określa się w momencie uruchomienia kontenera za pomocą polecenia run. Oto przykład takiego polecenia zmniejszającego o połowę priorytet procesora i ograniczającego dostępną pamięć do 128 MB:

```
$ docker container run -d --name nginx-test --cpu-shares 512 --memory 128M
-p 8080:80 nginx
```

Nasz przykładowy kontener jest już uruchomiony, a więc możemy spróbować skorzystać z polecenia update aktualizującego ograniczenia kontenera:

```
$ docker container update --cpu-shares 512 --memory 128M nginx-test
```

Uruchomienie tego polecenia spowoduje wyświetlenie komunikatu informującego o tym, że ograniczenie pamięci powinno być mniejsze od obecnego ograniczenia pliku wymiany (memoryswap):

```
Error response from daemon: Cannot update container e6ac3ce1418d520233a68f71a1e5cb3
→8b1ab0aafab5fa00d6e0220975706b246: Memory limit should be smaller than already set
→memoryswap limit, update the memoryswap at the same time
```

Jaki jest obecny limit pliku wymiany (memoryswap)? Możemy go określić za pomocą polecenia inspect wyświetlającego dane konfiguracji aktywnego kontenera. Spróbujmy uruchomić polecenie:

```
$ docker container inspect nginx-test
```

Dane konfiguracji są dość duże. Po uruchomieniu tego polecenia zwrócona została tablica JSON zajmująca 199 linii. Przefiltrujmy ją za pomocą polecenia grep w celu znalezienia linii zawierających słowo memory (pamięć):

```
$ docker container inspect nginx-test | grep -i memory
```

Polecenie to zwróci następujące dane:

```
"Memory": 0,
"KernelMemory": 0,
"MemoryReservation": 0,
"MemorySwap": 0,
"MemorySwappiness": -1,
```

Wszystkie parametry przyjmują wartość 0. Jak 128M może być mniejsze od 0? W kontekście konfiguracji zasobów 0 oznacza wartość domyślną, a więc brak ograniczeń (zauważ, że zazwyczaj nie umieszczono oznaczenia jednostki, takiego jak np. M). W związku z tym powinniśmy użyć następującego polecenia update:

```
$ docker container update --cpu-shares 512 --memory 128M --memory-swap 256M nginx-test
```

Stronicowanie pamięci to operacja wykonywana przez system operacyjny, w wyniku której część danych jest przenoszona z głównej pamięci do pamięci pomocniczej. Zabieg ten umożliwia rozszerzenie dostępnej pamięci fizycznej.

Domyślnie po dodaniu do polecenia run flagi --memory Docker przypisuje parametrowi --memory-swap wartość dwukrotnie większą od --memory.

Teraz po uruchomieniu polecenia docker container stats nginx-test powinieneś zobaczyć nowe limity:

| 1. russ (docker) | | | | | | | |
|------------------|-------|---------------------|-------|-------------------|-----------|------|--|
| CONTAINER | CPU % | MEM USAGE / LIMIT | MEM % | NET I/O | BLOCK I/O | PIDS | |
| e6ac3ce1418d | 0.00% | 1.852 MiB / 128 MiB | 1.45% | 7.64 kB / 2.34 kB | 0 B / 0 B | 2 | |

Zmiany będą również widoczne po ponownym uruchomieniu polecenia docker container inspect nginx-test | grep -i memory:

```
"Memory": 134217728,
"KernelMemory": 0,
"MemoryReservation": 0,
"MemorySwap": 268435456,
"MemorySwappiness": -1,
```

Wszystkie wartości wyświetlane po uruchomieniu polecenia docker container inspect są wyrażone w bajtach.

Stany kontenerów i pozostałe polecenia

W ostatniej części tego podrozdziału przyjrzymy się stanom, w których mogą znajdować się kontenery, a także opiszemy kilka poleceń przeznaczonych do obsługi kontenerów, o których jeszcze nie wspominaliśmy.

Po uruchomieniu polecenia docker container ls -a w oknie Terminala wyświetcone zostaną następujące dane:

| 3. russ (bash) | | | | | | | |
|----------------|-------|-------------------------|-------------------|---------------------------|----------------------|------------------|--|
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES | |
| 44e72e11b672 | nginx | "nginx -g 'daemon ...'" | About an hour ago | Exited (0) 24 seconds ago | 0.0.0.0:8080->80/tcp | nginx-foreground | |
| becaa6874eac | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up About a minute | | nginx-test | |

Na liście wymieniono dwa kontenery. Jeden z nich jest aktywny (status Up), a z drugiego wysłyszmy (status Exited). Zanim przejdziemy dalej, spróbujmy uruchomić jeszcze pięć kolejnych kontenerów. Aby zrobić to szybko, skorzystajmy z następującego polecenia:

```
$ for i in {1..5}; do docker container run -d --name nginx$(printf "%i" $i) nginx; done
```

Po uruchomieniu polecenia docker container ls -a powinieneś zobaczyć pięć nowych kontenerów o nazwach od nginx1 do nginx5:

| 3. russ (bash) | | | | | | | |
|----------------|-------|-------------------------|-------------------|--------------------------|----------------------|------------------|--|
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES | |
| 277a7efa462 | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 17 seconds | 80/tcp | nginx5 | |
| 5bbc78af29c8 | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 28 seconds | 80/tcp | nginx4 | |
| 194183e8569b | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 23 seconds | 80/tcp | nginx3 | |
| 922b78b1e409 | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 27 seconds | 80/tcp | nginx2 | |
| c92a0696e02c | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 30 seconds | 80/tcp | nginx1 | |
| 44e72e11b672 | nginx | "nginx -g 'daemon ...'" | About an hour ago | Exited (0) 4 minutes ago | 0.0.0.0:8080->80/tcp | nginx-foreground | |
| becaa6874eac | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 5 minutes | | nginx-test | |

Pause i unpause

Spróbujmy wstrzymać pracę kontenera nginx1. Możemy to zrobić za pomocą polecenia:

```
$ docker container pause nginx1
```

Teraz po uruchomieniu polecenia docker container ls okaże się, że status kontenera zmienił się z Up na Paused:

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|-------|-------------------------|-------------------|-----------------------|----------------------|------------|
| 277a7efaf462 | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 4 minutes | 80/tcp | nginx5 |
| 5bbc78af29c8 | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 4 minutes | 80/tcp | nginx4 |
| 194183e8569b | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 4 minutes | 80/tcp | nginx3 |
| 92b27b881e40 | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 4 minutes | 80/tcp | nginx2 |
| c92a0696e02c | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 4 minutes (Paused) | 80/tcp | nginx1 |
| becaa68f4eac | nginx | "nginx -g 'daemon ...'" | About an hour ago | Up 9 minutes | 0.0.0.0:8080->80/tcp | nginx-test |

Proces kontenera nginx1 nie został zakończony, a więc w celu wyświetlenia informacji o jego statusie nie musielibyśmy korzystać z flagi -a. Praca kontenera została wstrzymana za pomocą modułu cgroups. Proces uruchomiony w kontenerze nie wie, że został wstrzymany i że można go wznowić.

Więcej informacji na temat funkcji wstrzymywania obsługiwanej przez moduł cgroups znajdziesz w oficjalnej dokumentacji jądra systemu Linux: <https://www.kernel.org/doc/Documentation/cgroup-v1/freezer-subsystem.txt>.

Jak zapewne domyślasz się, w celu wznowienia pracy wstrzymanego kontenera należy skorzystać z polecenia unpause:

```
$ docker container unpause nginx1
```

Możliwość wstrzymania pracy kontenera przydaje się, gdy chcesz zamrozić stan kontenera, ponieważ jeden z uruchomionych kontenerów działa nieprawidłowo, chcesz przyjrzeć mu się później i nie chcesz, aby wadliwy kontener wpływał negatywnie na pracę pozostałych kontenerów.

Stop, start, restart i kill

W tej sekcji opiszemy polecenia stop, start, restart i kill. Z polecenia start korzystaliśmy w celu wznowienia pracy kontenera o statusie Exited. Polecenie stop działa tak samo jak wcisnięcie kombinacji klawiszy **Ctrl+C** w celu odłączenia pierwszoplanowego kontenera. Uruchom następujące polecenie:

```
$ docker container stop nginx2
```

Uruchomienie tego polecenia wysyła żądanie SIGTERM mające na celu zakończenie procesu. Jeżeli proces nie zakończy się sam w określonym czasie, to wysyłany jest sygnał SIGKILL, który powoduje natychmiastowe zakończenie procesu, nie dając mu czasu na zakończenie

wykonywania operacji powodującej opóźnienie (taką operacją może być np. zapis na dysku wyników zapytania wysłanego do bazy danych).

Siłowe zamknięcie procesu może mieć negatywne skutki. W związku z tym Docker umożliwia zmianę domyślnego czasu oczekiwania przed wysłaniem sygnału SIGKILL (domyślnie jest to 10 sekund). W celu zmiany tej wartości należy skorzystać z flagi `-t` (skrótu flagi `--time`). Oto przykładowe polecenie, które odczeka 60 sekund przed wysłaniem sygnału SIGKILL:

```
$ docker container stop -t 60 nginx3
```

Polecenie `start` może być użyte w celu ponownego uruchomienia procesu, ale w przeciwnieństwie do pary poleceń `pause` i `unpause` powoduje ono uruchomienie procesu od początku, a nie od punktu, w którym został wstrzymany.

```
$ docker container start nginx2 nginx3
```

Polecenie `restart` jest kombinacją dwóch poleceń — zatrzymuje, a następnie ponownie uruchamia kontener określony poprzez przekazanie identyfikatora lub nazwy. Polecenie to, podobnie jak polecenie `stop`, obsługuje flagę `-t`:

```
$ docker container restart -t 60 nginx4
```

Istnieje jeszcze możliwość natychmiastowego wysłania do kontenera sygnału SIGKILL. W tym celu należy skorzystać z polecenia `kill`:

```
$ docker container kill nginx5
```

Usuwanie kontenerów

Przyjrzyjmy się kontenerom, które uruchomiliśmy — skorzystajmy z polecenia `docker container ls -a`. Po uruchomieniu tego polecenia na moim komputerze okazało się, że dwa kontenery mają status `Exited`, a wszystkie pozostałe są aktywne:

| 3. russ (bash) | | | | | | |
|----------------|-------|------------------------|--------------|----------------------------|----------------------|------------------|
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
| 277a7efaf462 | nginx | "nginx -g 'daemon ...' | 16 hours ago | Exited (137) 5 minutes ago | | nginx5 |
| 5bbc78af29c8 | nginx | "nginx -g 'daemon ...' | 16 hours ago | Up 5 minutes | 80/tcp | nginx4 |
| 194183e8569b | nginx | "nginx -g 'daemon ...' | 16 hours ago | Up 14 minutes | 80/tcp | nginx3 |
| 922b7b881e49 | nginx | "nginx -g 'daemon ...' | 16 hours ago | Up 2 hours | 80/tcp | nginx2 |
| c92a0696e92c | nginx | "nginx -g 'daemon ...' | 16 hours ago | Up 2 hours | 80/tcp | nginx1 |
| 44e7e11b672 | nginx | "nginx -g 'daemon ...' | 16 hours ago | Exited (8) 2 hours ago | 0.0.0.0:8080->80/tcp | nginx-foreground |
| becaa68f4eac | nginx | "nginx -g 'daemon ...' | 16 hours ago | Up 2 hours | | nginx-test |

W celu usunięcia dwóch kontenerów mających status `Exited` mogę skorzystać z polecenia `prune`:

```
$ docker container prune
```

Po uruchomieniu tego polecenia zostałem zapytany o potwierdzenie chęci skasowania kontenerów:

```
4. russ (bash)
⚡ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
277a7efaf42e83675105e5d39c8c00b34fcf31ed707646d4ded8939b31b603b
44e72e11b672a7453f158588f37dbd5c92c00a66c115320d594cba5e1909da22

Total reclaimed space: 2 B
russ in ~
⚡
```

W celu wybrania kontenera do skasowania można skorzystać z polecenia `rm`. Oto przykład jego użycia:

```
$ docker container rm nginx4
```

Próba skasowania aktywnego kontenera spowoduje wyświetlenie następującego komunikatu:

```
Error response from daemon: You cannot remove a running container 5bbc78af29c87
↳ 1200539e7534725e12691aa7df6facd616e91acbe41f204b734. Stop the container before
↳ attempting removal or use -f
```

Komunikat ten informuje o tym, że w celu usunięcia aktywnego kontenera należy skorzystać z flagi `-f`, która zatrzyma kontener, a następnie go usunie. W takim przypadku należy skorzystać z następującego polecenia:

```
$ docker container rm -f nginx4
```

Alternatywnym rozwiązaniem jest połączenie ze sobą polecień `stop` i `rm`:

```
$ docker container stop nginx3 && docker container rm nginx3
```

Na razie nie obchodzi nas zbytnio poprawność zamknięcia procesów i chcemy po prostu usunąć kontenery, a więc wpisywanie tak długiego polecenia można uznać za przerost formy nad treścią (zwłaszcza dlatego, że możemy teraz użyć polecenia `prune`).

Możesz usunąć pozostałe kontenery, korzystając z dowolnej metody.

Inne polecenia

W ostatniej sekcji tego podrozdziału przyjrzymy się kilku poleceniom, z których prawdopodobnie nie będziesz korzystać podczas codziennej pracy z Dockerem. Zaczniemy od polecenia `create`. Działa ono podobnie jak polecenie `run`, ale zamiast uruchamiać kontener, przygotowuje ono kontener do pracy i go konfiguruje:

```
$ docker container create --name nginx-test -p 8080:80 nginx
```

Status utworzonego kontenera możemy sprawdzić za pomocą polecenia `docker container ls -a`. Przed kolejnym sprawdzeniem statusu kontenera uruchomimy go za pomocą polecenia `docker container run nginx-test`:

```

3. russ (bash)
⚡ docker container create --name nginx-test -p 8080:80 nginx
011111f49f43f2eb3779e3d39aa9538e6092a189533b35594a84a7db78b70f
russ in ~
⚡ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
011111f49f43        nginx              "nginx -g 'daemon ...
russ in ~
⚡ docker container start nginx-test
nginx-test
russ in ~
⚡ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
011111f49f43        nginx              "nginx -g 'daemon ...
russ in ~
⚡

```

Teraz przyjrzymy się poleceniu port. Wyświetla ono informacje o mapowaniu portów kontenera:

```
$ docker container port nginx-test
```

Polecenie to powinno zwrócić następującą informację:

```
80/tcp -> 0.0.0.0:8080
```

Informacje o portach są również wyświetlane przez polecenie docker container ls.

Na koniec przyjrzyjmy się poleceniu diff. Wyświetla ono listę wszystkich plików, które zostały zmodyfikowane lub dodane do kontenera od momentu uruchomienia kontenera, a więc pozwala na określenie różnic pomiędzy systemem pliku obrazu użytego do uruchomienia kontenera a obecnym stanem kontenera.

Zanim uruchomimy to polecenie, utworzymy pusty plik w kontenerze nginx-test. Zróbmy to za pomocą polecenia exec:

```
$ docker container exec nginx-test touch /tmp/test
```

Polecenie to utworzyło plik o nazwie *test* w katalogu /tmp. Teraz możemy sprawdzić różnice pomiędzy oryginalnym obrazem i działającym kontenerem:

```
$ docker container diff nginx-test
```

Polecenie to zwróci listę plików. Jak widzisz, na liście znalazły się pliki o nazwie *test*, a także inne pliki utworzone od momentu uruchomienia kontenera:

```
C /run
A /run/nginx.pid
C /tmp
A /tmp/test
C /var/cache/nginx
A /var/cache/nginx/client_temp
A /var/cache/nginx/fastcgi_temp
A /var/cache/nginx/proxy_temp
A /var/cache/nginx/scgi_temp
A /var/cache/nginx/uwsgi_temp
```

Warto zauważyc, że po zatrzymaniu i usunięciu kontenera pliki te zostaną utracone. W kolejnym podrozdziale opiszemy zagadnienia związane z wolumenami Dockera, które pozwalają na zachowanie danych wygenerowanych podczas pracy kontenera.

Jeżeli wykonujesz opisywane przez nas operacje, to usuń wszystkie kontenery uruchomione podczas lektury tego podrozdziału (możesz to zrobić za pomocą dowolnej metody).

Obsługa sieci i wolumenów

W tym podrozdziale przedstawimy podstawowe zagadnienia związane z obsługą sieci i wolumenów za pomocą podstawowych sterowników. Zaczniemy od obsługi sieci.

Sieć

Dotychczas uruchamialiśmy kontenery w pojedynczej, płaskiej i współdzielonej sieci. W związku z tym uruchamiane dotychczas kontenery mogły komunikować się ze sobą bez konieczności korzystania z funkcji sieciowych hosta.

Przeanalizujmy przykład zamiast zagłębiać się w zbędne szczegóły. Uruchomimy aplikację składającą się z dwóch kontenerów. W pierwszym kontenerze będzie działała baza Redis, a w drugim aplikacja zapisująca stan systemu w bazie Redis.

Redis jest strukturą danych, która może być używana jako baza danych, pamięć podręczna lub miejsce zapisu komunikatów. Obsługuje ona różne poziomy trwałości zapisu. Więcej informacji na temat Redis znajdziesz na stronie <https://redis.io>.

Przed uruchomieniem aplikacji musimy pobrać obrazy, z których będziemy korzystać, a także utworzyć sieć:

```
$ docker image pull redis:alpine  
$ docker image pull russmckendrick/moby-counter  
$ docker network create moby-counter
```

Po uruchomieniu tych poleceń w oknie Terminala zobaczysz komunikaty podobne do następujących (patrz pierwszy zrzut na następnej stronie).

Po pobraniu obrazów i utworzeniu sieci możemy uruchomić kontenery. Zaczniemy od kontenera Redis:

```
$ docker container run -d --name redis --network moby-counter redis:alpine
```

Polecenie to zawierało flagę --network definiującą sieć, w której ma zostać uruchomiony kontener. Po uruchomieniu kontenera Redis możemy uruchomić kontener aplikacji:

```
$ docker container run -d --name moby-counter --network moby-counter -p  
8080:80 russmckendrick/moby-counter
```

```
1. moby-counter (bash)
russ in ~/Documents/Code/docker/moby-counter on master*
⚡ docker image pull redis:alpine
alpine: Pulling from library/redis
43d680a959df: Pull complete
4371af8a8c8e: Pull complete
99ed756629db: Pull complete
653720cb4fd4: Pull complete
a65f285e1862: Pull complete
f0d06b31f9ca: Pull complete
Digest: sha256:abd220f66aac9f530096d5c4d382eb1483dc4e3bac5dd6a56b67746cd86d0e5b
Status: Downloaded newer image for redis:alpine
russ in ~/Documents/Code/docker/moby-counter on master*
⚡ docker image pull russmckendrick/moby-counter
Using default tag: latest
latest: Pulling from russmckendrick/moby-counter
cfc728c1c558: Pull complete
21552fe8e0cd: Pull complete
3025e3fe6722: Pull complete
5c51b5fc92b9: Pull complete
da1cc31641fc: Pull complete
0477dda24b67: Pull complete
Digest: sha256:69dd4b4ae6ef09d6e87cc7bd1b0f3a71d083b844f60d20ddbdefba9872dcef80
Status: Downloaded newer image for russmckendrick/moby-counter:latest
russ in ~/Documents/Code/docker/moby-counter on master*
⚡ docker network create moby-counter
adfd04f86ceb0e0138b3c46c5ffac98810a572895583af0a5fce644378ce16c8
russ in ~/Documents/Code/docker/moby-counter on master*
⚡ 
```

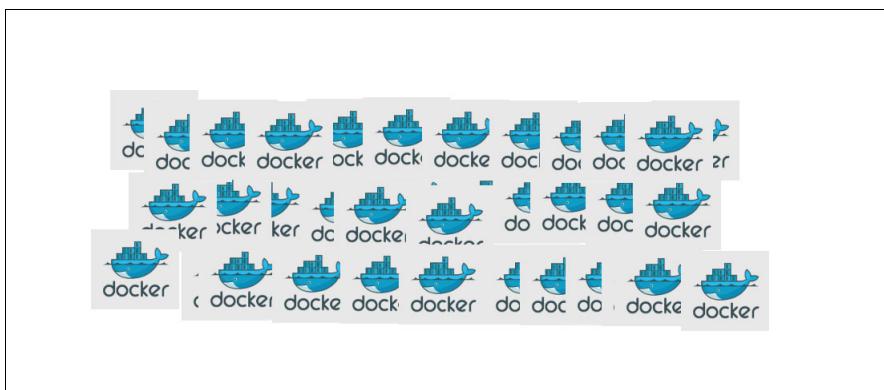
Kontener został uruchomiony w sieci moby-counter, a port 80 kontenera jest mapowany do portu 8080 hosta. Zauważ, że nie musieliszy się martwić odsłanianiem portów kontenera Redis, ponieważ kontener ten automatycznie odsłania domyślny port (6379). Można to sprawdzić, uruchamiając polecenie docker container ls:

```
3. russ (bash)
⚡ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
82d5183ded1        russmckendrick/moby-counter   "node index.js"      14 hours ago       Up 12 seconds      0.0.0.0:8080->80/tcp   moby-counter
1bccc138459a        redis:alpine          "docker-entrypoint..."  14 hours ago       Up 19 seconds      6379/tcp            redis
russ in ~
⚡ 
```

Teraz musimy tylko uzyskać dostęp do aplikacji. W tym celu należy uruchomić przeglądarkę internetową i wejść pod adres <http://localhost:8080/>. Na ekranie powinna pojawić się pusta strona z komunikatem *Click to add logos* (kliknij, aby dodać logo):



Kliknięcie dowolnego miejsca strony spowoduje umieszczenie w nim logo Dockera:



Jakie operacje są wykonywane w tej chwili? Aplikacja uruchomiona w kontenerze `moby-counter` łączy się z kontenerem `redis` i korzysta z bazy Redis w celu zapisania w niej współrzędnych, w których umieszczone zostały obrazy (wykonano kliknięcie).

Jak aplikacja `moby-counter` łączy się z kontenerem `redis`? W pliku `server.js` znajdują się następujące domyślne wartości:

```
var port = opts.redis_port || process.env.USE_REDIS_PORT || 6379  
var host = opts.redis_host || process.env.USE_REDIS_HOST || 'redis'
```

Wynika z nich, że aplikacja `moby-counter` stara się nawiązać połaczenie z hostem o nazwie `redis` za pośrednictwem portu 6379. Spróbujmy skorzystać z polecenia `exec` w celu przeprowadzenia operacji pingowania kontenera `redis` z kontenera aplikacji `moby-counter`:

```
$ docker container exec moby-counter ping -c 3 redis
```

Na ekranie powinny pojawić się dane typu:

```
PING redis (172.18.0.2): 56 data bytes  
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.057 ms  
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.087 ms  
64 bytes from 172.18.0.2: seq=2 ttl=64 time=0.085 ms  
  
--- redis ping statistics ---  
3 packets transmitted, 3 packets received, 0% packet loss  
round-trip min/avg/max = 0.057/0.076/0.087 ms
```

Jak widzisz, kontener `moby-counter` określa adres IP hosta `redis` — kontener bazy Redis znajduje się pod adresem 172.18.0.2. Może Ci się wydawać, że wynika to z tego, że plik `hosts` aplikacji zawiera adres IP kontenera `redis`. Sprawdź to za pomocą następującego polecenia:

```
$ docker container exec moby-counter cat /etc/hosts
```

Polecenie to zwraca zawartość pliku */etc/hosts*, która w przypadku mojego kontenera wygląda następująco:

```
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.3 8ed5183eded1
```

Na końcu pliku znajduje się wpis definiujący nazwę hosta lokalnego kontenera — 8ed5183eded1 jest identyfikatorem kontenera. Jak widzisz, plik ten nie zawiera żadnego elementu definiującego adres IP kontenera *redis*. Sprawdź zawartość pliku */etc/resolv.conf*. W tym celu uruchom polecenie:

```
$ docker container exec moby-counter cat /etc/resolv.conf
```

Tutaj znajdziemy to, czego szukaliśmy. Okazuje się, że korzystamy z lokalnego serwera nazw (nameserver):

```
nameserver 127.0.0.11
options ndots:0
```

Spróbujmy ustalić adres IP kontenera o nazwie *redis*, korzystając z serwera DNS działającego pod adresem 127.0.0.11:

```
$ docker container exec moby-counter nslookup redis 127.0.0.11
```

Polecenie zwróci adres IP kontenera *redis*:

```
Server: 127.0.0.11
Address 1: 127.0.0.11

Name: redis.moby-counter
Address 1: 172.18.0.2 redis.moby-counter
```

Utwórzmy drugą sieć i uruchommy kolejny kontener aplikacji:

```
$ docker network create moby-counter2
$ docker run -itd --name moby-counter2 --network moby-counter2 -p 9090:80
russmckendrick/moby-counter
```

Po uruchomieniu drugiej aplikacji spróbujmy wykonać z jej poziomu operację pingowania kontenera *redis*:

```
$ docker container exec moby-counter2 ping -c 3 redis
```

Uzyskałem odpowiedź, ale nie jest to odpowiedź, jakiej się spodziewałem:

```
PING redis (92.242.132.15): 56 data bytes
64 bytes from 92.242.132.15: seq=0 ttl=37 time=0.314 ms
64 bytes from 92.242.132.15: seq=1 ttl=37 time=0.211 ms
```

```
64 bytes from 92.242.132.15: seq=2 ttl=37 time=0.233 ms
```

```
--- redis ping statistics ---3 packets transmitted, 3 packets received, 0% packet loss  
round-trip min/avg/max = 0.211/0.252/0.314 ms
```

Po zjazdzeniu do pliku *resolv.conf* okazało się, że używany był ten sam serwer nazw (nameserver):

```
$ docker container exec moby-counter2 cat /etc/resolv.conf  
  
nameserver 127.0.0.11  
options ndots:0
```

Próba określenia adresu hosta o nazwie *redis* kończy się niepowodzeniem — kontener próbuje odwoływać się do domyślnych serwerów DNS firmy zapewniającej mi połączenie internetowe, co zasadniczo kończy się zwrotniem błędem:

```
$ docker container exec moby-counter2 nslookup redis 127.0.0.11  
Server: 127.0.0.11  
Address 1: 127.0.0.11  
  
Name: redis  
Address 1: 92.242.132.15 unallocated.barefruit.co.uk
```

Przyjrzymy się procesowi uruchamiania drugiego serwera Redis. Uruchommy go w drugiej sieci. Jak wiesz, nie można uruchomić dwóch kontenerów o tej samej nazwie, a więc wykażmy się kreatywnością i nadajmy mu nazwę *redis2*.

Aplikacja jest skonfigurowana tak, aby określać adres serwera o nazwie *redis*. Czy w związku z tym musimy modyfikować kontener z aplikacją? Nie. Pomoże nam Docker.

Nie możemy uruchomić dwóch kontenerów o takich samych nazwach, a druga sieć jest całkowicie odizolowana od pierwszej, a więc możemy wciąż korzystać z nazwy DNS *redis*. Aby skorzystać z tej możliwości, musimy skorzystać z flagi *--network-alias*:

```
$ docker container run -d --name redis2 --network moby-counter2 --network-alias  
↳redis redis:alpine
```

Mamy kontener o nazwie *redis2*, któremu za pomocą flagi *--network-alias* przypisaliśmy alias *redis*, a więc możemy spróbować ustalić jego adres IP:

```
$ docker container exec moby-counter2 nslookup redis 127.0.0.11  
Server: 127.0.0.11  
Address 1: 127.0.0.11  
  
Name: redis  
Address 1: 172.19.0.3 redis2.moby-counter2
```

Jak widzisz, *redis* jest aliasem kontenera *redis2.moby-counter2*, któremu przypisano adres IP 172.19.0.3.

Teraz mamy dwie aplikacje uruchomione równolegle we własnych odizolowanych sieciach. Dostęp do nich można uzyskać, korzystając z adresów <http://localhost:8080/> i <http://localhost:9090/>. Po uruchomieniu polecenia docker network ls zobaczysz listę sieci uruchomionych w lokalnym hoście Dockera (lista ta zawiera również domyślne sieci):

```
⚡ docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
732497f540d5    bridge    bridge      local
89729940d70f    host      host       local
adfd04f86ceb    moby-counter    bridge      local
f4ea6b65bb8e    moby-counter2   bridge      local
ec7b8ac4791c    none      null       local
russ in ~/Documents/Code/docker/moby-counter on master*
⚡
```

Więcej informacji na temat konfiguracji sieci możesz znaleźć, uruchamiając polecenie inspect:

```
$ docker network inspect moby-counter
```

Polecenie to zwróci następującą tablicę JSON:

```
[
  {
    "Name": "moby-counter",
    "Id": "adfd04f86ceb0e0138b3c46c5ffac98810a572895583af0a5fce644378ce16c8",
    "Created": "2017-06-24T21:45:59.109846671Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Containers": {
      "1becce530450ab46450d185be2def 9d7820ba3da36948cb77f5b51a931bb58738": {
        "Name": "redis",
        "EndpointID": "6f5199ec31cd852f7eff961fa9afdf94810c602af69d5a870f6209bbaed087b56",
        "MacAddress": "02:42:ac:12:00:02",
        "IPv4Address": "172.18.0.2/16",
        "IPv6Address": ""
      },
      "8ed5183edded1c904b86c279ab15ac44ffe15c2a57f1fb2287170f221dc320633": {
        "Name": "moby-counter",
        "EndpointID": "
```

```
        "fcad0cbb7249b355bead29d424919f6549c111a3a0364116d2d6e1d02b217411",
        "MacAddress": "02:42:ac:12:00:03",
        "IPv4Address": "172.18.0.3/16",
        "IPv6Address": ""
    }
},
"Options": {},
"Labels": {}
]
}
```

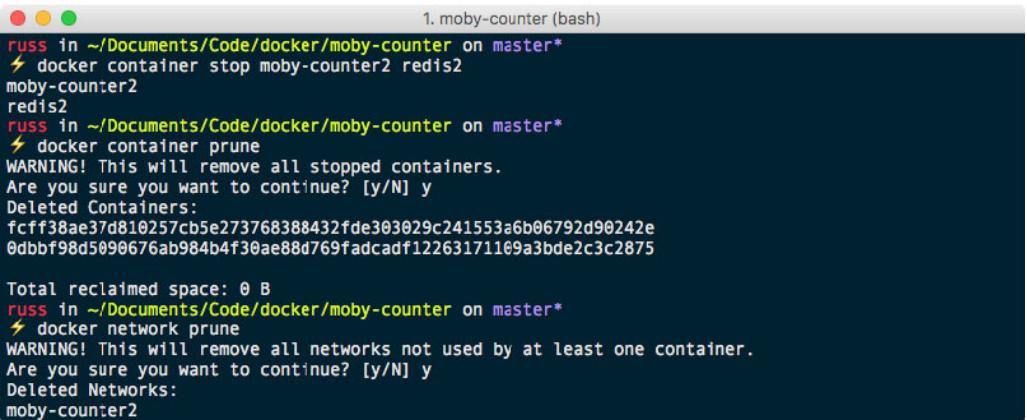
Tablica ta w sekcji IPAM zawiera informacje dotyczące adresowania sieciowego. Ponadto znajdziesz w niej również informacje o obu kontenerach aktywnych w sieci.

IPAM (zarządzanie adresami IP) jest systemem planowania, śledzenia i zarządzania przestrzenią adresową protokołu internetowego używanego do obsługi sieci. System ten zawiera narzędzia DNS i DHCP, dzięki czemu usługi są powiadamiane o modyfikacji innych usług — np. podczas przypisywania adresu IP kontenerowi kontener2 przez usługę DHCP dochodzi do automatycznej aktualizacji usługi DNS tak, aby zwracała ona właściwy adres tego kontenera.

Przed przejściem do kolejnej sekcji powinniśmy usunąć jedną z aplikacji i związaną z nią sieć. Zróbjmy to za pomocą następujących poleceń:

```
$ docker container stop moby-counter2 redis2
$ docker container prune
$ docker network prune
```

Polecenia te usuną kontenery i sieć:



```
1. moby-counter (bash)
russ in ~/Documents/Code/docker/moby-counter on master*
⚡ docker container stop moby-counter2 redis2
moby-counter2
redis2
russ in ~/Documents/Code/docker/moby-counter on master*
⚡ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
fcff38ae37d810257cb5e273768388432fde303029c241553a6b06792d90242e
0dbbf98d5090676ab984b4f30ae88d769fadcadf12263171109a3bde2c3c2875

Total reclaimed space: 0 B
russ in ~/Documents/Code/docker/moby-counter on master*
⚡ docker network prune
WARNING! This will remove all networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
moby-counter2
```

W tej sekcji korzystaliśmy tylko z domyślnego sterownika sieciowego — ograniczyliśmy się tylko do tworzenia sieci w obrębie jednego hosta Dockera. W kolejnych rozdziałach zajmiemy się rozszerzaniem sieci Dockera i używaniem jej do komunikacji pomiędzy wieloma hostami Dockera, a nawet wieloma dostawcami usługi dostępu do internetu.

Wolumeny Dockera

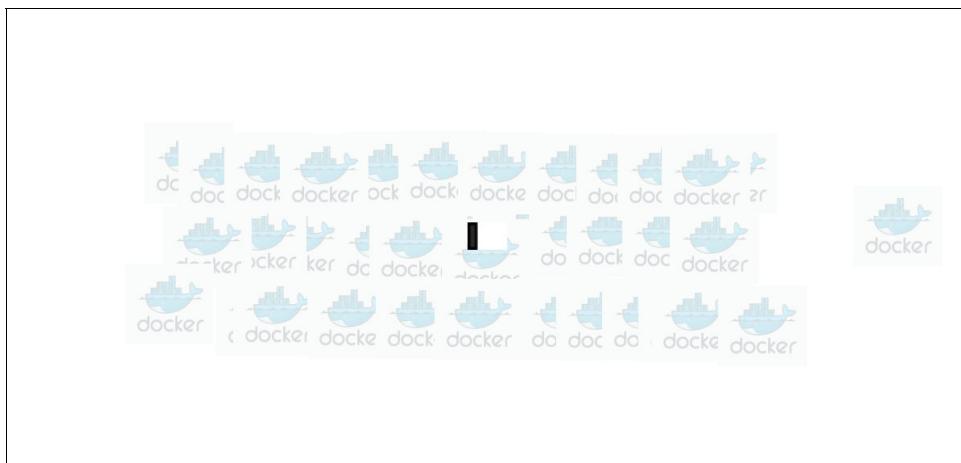
Jeżeli wykonałeś wszystkie polecenia przedstawione w poprzedniej sekcji, to powinieneś dysponować dwoma aktywnymi kontenerami:

```
3. russ (bash)
$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
8ed5183ded1        russmckendrick/moby-counter   "node index.js"   15 hours ago      Up 15 hours        0.0.0.0:8080->80/tcp   moby-counter
1bece530450a       redis:alpine          "docker-entrypoint..." 15 hours ago      Up 15 hours        6379/tcp           redis
russ 1n ~
$
```

Jeżeli otworzysz przeglądarkę internetową i wejdziesz na stronę <http://localhost:8080/>, to zobaczyś umieszczone tam wcześniej logo Dockera. Zatrzymajmy i usuńmy kontener Redis i zobaczymy, co się stanie. W tym celu należy uruchomić polecenia:

```
$ docker container stop redis
$ docker container rm redis
```

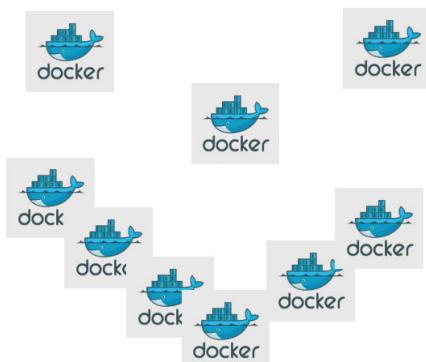
Jeżeli nie zamknąłeś okna przeglądarki, to mogłeś zauważyc znikające logo Dockera i animowany symbol ładowania wyświetlony na środku ekranu. Symbol ten oznacza czekanie aplikacji na ponowne nawiązanie połączenia z kontenerem Redis:



Czas uruchomić ponownie kontener Redis za pomocą następującego polecenia:

```
$ docker container run -d --name redis --network moby-counter redis:alpine
```

Połączenie z serwerem zostanie przywrócone, ale na ekranie nie pojawią się umieszczone wcześniej logo. Aplikacja wróci do stanu początkowego. Kliknij kilka razy, dodając logo Dockera. Umieść je w innych miejscach niż wcześniej. Ja zrobiłem to w następujący sposób:



Po dodaniu nowej kombinacji symboli Dockera ponownie usuńmy kontener Redis:

```
$ docker container stop redis  
$ docker container rm redis
```

Przypominam, że usunięcie kontenera wiąże się z utratą danych, ale jeżeli korzystałeś z oficjalnego obrazu Redis pobranego ze strony <https://store.docker.com/images/redis/>, to tak naprawdę nie straciłeś żadnych danych.

Oto plik Dockerfile oficjalnego obrazu serwera Redis:

```
FROM alpine:3.5

RUN addgroup -S redis && adduser -S -G redis redis
RUN apk add --no-cache 'su-exec>=0.2'

ENV REDIS_VERSION 3.0.7
ENV REDIS_DOWNLOAD_URL http://download.redis.io/releases/redis-3.0.7.tar.gz
ENV REDIS_DOWNLOAD_SHA e56b4b7e033ae8dbf311f9191cf6fdf3ae974d1c

RUN set -x \  

    && apk add --no-cache --virtual .build-deps \  

        gcc \  

        linux-headers \  

        make \  

        musl-dev \  

        tar \  

    && wget -O redis.tar.gz "$REDIS_DOWNLOAD_URL" \  

    && echo "$REDIS_DOWNLOAD_SHA *redis.tar.gz" | shasum -c - \  

    && mkdir -p /usr/src/redis \  

    && tar -xzf redis.tar.gz -C /usr/src/redis --strip-components=1 \  

    && rm redis.tar.gz \  

    && make -C /usr/src/redis \  


```

```

&& make -C /usr/src/redis install \
&& rm -r /usr/src/redis \
&& apk del .build-deps

RUN mkdir /data && chown redis:redis /data
VOLUME /data
WORKDIR /data

COPY docker-entrypoint.sh /usr/local/bin/
RUN ln -s usr/local/bin/docker-entrypoint.sh /entrypoint.sh # wsteczna kompatybilność
ENTRYPOINT ["docker-entrypoint.sh"]

EXPOSE 6379
CMD [ "redis-server" ]

```

Zwróć uwagę na umieszczone bliżej końca pliku dyrektywy VOLUME i WORKDIR. Dzięki nim Docker w momencie uruchamiania kontenera tworzy wolumen, a następnie uruchamia z niego usługę redis-server.

Możemy do sprawdzić, uruchamiając polecenie:

```
$ docker volume ls
```

Na wyświetlanej przez nie liście powinny znaleźć się przynajmniej dwa wolumeny:

```

1. russ (bash)
$ docker volume ls
DRIVER    VOLUME NAME
local     719d0cc415dbc76fed5e9b8893e2cf547f0ac6c91233451604fdb31f0dd2d2a
          d6107f7ee25b55011f1f144e9030ee44140b463c20925450bf60065e78fee54b
russ     in ~
$ 

```

Jak widzisz, nazwa wolumenu nie jest zbyt przyjazna. Jest ona tak naprawdę jego unikalnym identyfikatorem. Jak można korzystać z wolumenu po uruchomieniu kontenera Redis? Z pliku Dockerfile wynika, że wolumen został zamontowany wewnątrz kontenera w folderze `/data` (dane), a więc wystarczy poinformować Dockera o tym, którego wolumenu ma użyć i gdzie ma go zamontować.

W tym celu należy skorzystać z następującego polecenia (skorzystać z identyfikatora swojego wolumenu):

```

$ docker container run -d --name redis -v
719d0cc415dbc76fed5e9b8893e2cf547f0ac6c91233451604fdb31f0dd2d2a:/data --network
  ↵moby-counter redis:alpine

```

Jeżeli strona aplikacji wyświetlona w przeglądarce wygląda tak, jakby aplikacja ponownie starała się nawiązać połączenie z serwerem Redis, to spróbuj odświeżyć stronę w przeglądarce. Jeżeli to nie pomoże, to musisz uruchomić ponownie kontener, korzystając z polecenia `docker container restart moby-counter`, i ponownie odświeżyć stronę wyświetlana w przeglądarce.

Powinieneś zobaczyć ikony Dockera w miejscach, w których je umieściłeś na początku. W celu podejrzenia zawartości folderu `/data` kontenera serwera Redis należy skorzystać z polecenia:

```
$ docker container exec redis ls -lhat /data
```

Spowoduje ono zwrócenie listy plików:

```
total 12
drwxr-xr-x 1 root root 4.0K Jun 25 14:55 ..
drwxr-xr-x 2 redis redis 4.0K Jun 25 14:00 .
-rw-r--r-- 1 redis redis 421 Jun 25 14:00 dump.rdb
```

Możesz usunąć aktywny kontener, a następnie uruchomić go ponownie, korzystając z identyfikatora drugiego volumenu.

Możesz również skorzystać z volumenu wygenerowanego samodzielnie. W tym celu należy posłużyć się poleceniem `volume`:

```
$ docker volume create redis_data
```

Po utworzeniu volumenu `redis_data` możesz go używać do przechowywania danych bazy Redis — wystarczy, że uruchomisz polecenie:

```
$ docker container run -d --name redis -v redis_data:/data --network moby-counter
→redis:alpine
```

Po uruchomieniu tego polecenia możliwe jest korzystanie z utworzonego wcześniej volumenu:

```
russ in ~
⚡ docker volume create redis_data
redis_data
russ in ~
⚡ docker container run -d --name redis -v redis_data:/data --network moby-counter redis:alpine
fbe37a9162841alec04fe38ffc41461d63a73c0cf2aa7b4d38d071fcdb98c35b
russ in ~
⚡ docker container restart moby-counter
moby-counter
russ in ~
⚡ docker container rm -f redis
redis
russ in ~
⚡ docker container run -d --name redis -v redis_data:/data --network moby-counter redis:alpine
177937e2f058bb424d7b461b4257fd09e41f61f1465c4b0b0988192b005a0f65
russ in ~
⚡
```

Podobnie jak jest w przypadku polecenia `network`, w celu wyświetlenia bardziej szczegółowych informacji na temat kontenera należy skorzystać z polecenia `inspect`:

```
$ docker volume inspect redis_data
```

Oto zwrócona przez nie tablica danych:

```
[{"Driver": "local", "Labels": {}, "Mountpoint": "/var/lib/docker/volumes/redis_data/_data"},
```

```

    "Name": "redis_data",
    "Options": {},
    "Scope": "local"
}
]

```

Jak widzisz, używanie wolumenu podczas pracy z lokalnym sterownikiem nie jest niczym skomplikowanym. Warto zwrócić uwagę, że dane przechowywane są w folderze `/var/lib/docker/volumes/redis_data/_data` hosta Dockera. Jeżeli pracujesz w systemie macOS lub Windows, to dane te będą przechowywane w maszynie wirtualnej hosta Dockera, a nie w lokalnym systemie plików, a więc nie będziesz mógł uzyskać do nich bezpośredniego dostępu.

Na razie się tym nie przejmuj. W kolejnych rozdziałach opiszymy dokładniej zagadnienia związane z korzystaniem z danych zgromadzonych w wolumenach Dockera. Na razie musimy oczyścić środowisko, w którym pracujemy. Zaczniemy od usunięcia dwóch kontenerów i sieci:

```

$ docker container stop redis moby-counter
$ docker container prune
$ docker network prune

```

Teraz możemy usunąć wolumeny za pomocą polecenia:

```
$ docker volume prune
```

W oknie Terminala powinny zostać wyświetlane następujące informacje:

```

russ in ~
$ docker container stop redis moby-counter
redis
moby-counter
russ in ~
$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
177937e2f058bb424d7b461b4257fd09e41f61f1465c4b0b0988192b005a0f65
8ed5183eeded1c904b86c279ab15ac4ffe15c2a57f1fb2287170f221dc320633

Total reclaimed space: 0 B
russ in ~
$ docker network prune
WARNING! This will remove all networks not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Networks:
moby-counter

russ in ~
$ docker volume prune
WARNING! This will remove all volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
71900cc415dbc76fed5e9b8893e2cf547f0ac6c91233451604fdb31f0dd2d2a
d6107f7ee25b55011f1f144e9030ee44140b463c20925450bf60065e78fee54b
redis_data

Total reclaimed space: 743 B
russ in ~
$ 

```

Nasze środowisko pracy wróciło do stanu początkowego.

Podsumowanie

W tym rozdziale poruszyliśmy zagadnienia związane z zarządzaniem z poziomu wiersza poleceń pojedynczymi kontenerami, a także aplikacjami korzystającymi z wielu kontenerów pracującymi w odizolowanych sieciach Docker'a. Ponadto opisaliśmy również proces zapisywania w wolumenach Docker'a danych wygenerowanych przez kontenery.

Dotychczas opisaliśmy większość zagadnień związanych z poleceniami:

```
$ docker container [polecenie]  
$ docker network [polecenie]  
$ docker volume [polecenie]  
$ docker image [polecenie]
```

Zajęliśmy się czterema głównymi aspektami lokalnej pracy z Dockerem. Teraz możemy przejść do zagadnień związanych z pracą ze zdalnymi hostami.

W kolejnym rozdziale zajmiemy się narzędziem Docker Machine i zgłębimy kolejne zagadnienia związane z sieciowymi możliwościami Docker'a.

Docker Machine

W tym rozdziale przyjrzymy się narzędziu Docker Machine. Jest ono przeznaczone do uruchamiania i przygotowywania do pracy hostów Dockera na różnych platformach — w środowisku lokalnym i w chmurze. Umożliwia ono również sterowanie pracą hostów Dockera. Oto zagadnienia, które zostaną poruszone w tym rozdziale:

- Wprowadzenie do narzędzia Docker Machine.
- Używanie Docker Machine w celu uruchamiania hostów Dockera w środowisku lokalnym.
- Uruchamianie hostów Dockera w chmurze.
- Poszerzenie wiadomości dotyczących sieciowych możliwości Dockera.
- Korzystanie z bazowych systemów operacyjnych.

Wprowadzenie do narzędzia Docker Machine

Zanim zagłębimy się w szczegóły pracy narzędzia Docker Machine, powinniśmy umiejscowić je w ekosystemie Dockera.

Największą zaletą narzędzia Docker Machine jest to, że obsługuje wiele publicznych chmur:

- Amazon Web Services: <https://aws.amazon.com/ecr/>,
- Microsoft Azure: <https://azure.microsoft.com>,
- DigitalOcean: <https://www.digitalocean.com>,
- Exoscale: <https://www.exoscale.ch>,

- Google Compute Engine: <https://cloud.google.com>,
- Rackspace: <https://www.rackspace.com>,
- IBM SoftLayer: <https://www.softlayer.com>.

Docker Machine obsługuje również samodzielne maszyny wirtualne — platformy takie jak:

- Microsoft Hyper-V: <https://www.microsoft.com/en-gb/cloud-platform/server-virtualization/>,
- OpenStack: <https://www.openstack.org>,
- VMware vSphere: <https://www.vmware.com/uk/products/vsphere.html>.

Ponadto obsługiwani są również następujący lokalni hipernadzorcy:

- Oracle VirtualBox: <https://www.virtualbox.org>,
- VMware Fusion: <https://www.vmware.com/uk/products/fusion.html>,
- VMware Workstation: <https://www.vmware.com/uk/products/workstation.html>.

Możliwość korzystania z wszystkich tych technologii za pomocą jednego polecenia przy minimalnej interakcji użytkownika pozwala zaoszczędzić dużo czasu np. w przypadku potrzeby skorzystania z usług Amazon Web Services jednego dnia i DigitalOcean dnia kolejnego — dzięki Docker Machine obsługa obu tych rozwiązań wygląda tak samo.

Narzędzie Docker Machine jest obsługiwane z wiersza poleceń, co ułatwia przekazywanie instrukcji kolegom i tworzenie skryptów uruchamiających i zamykających hosty Dockera — wyobraź sobie możliwość budowania świeżego środowiska roboczego z rana i kasowania go wieczorem w celu zmniejszenia kosztów.

Docker Machine i wdrażanie lokalnych hostów Dockera

Zacznijemy od podstaw pracy z lokalnymi środowiskami (skorzystamy z maszyny wirtualnej VirtualBox), a później przejdziemy do pracy w chmurze.

W celu uruchomienia maszyny należy wywołać następujące polecenie:

```
$ docker-machine create --driver virtualbox docker-local
```

Polecenie to uruchomi wdrożenie, podczas którego uzyskasz listę zadań wykonywanych przez aktywną maszynę Docker Machine. Każdy z uruchamianych hostów musi przejść te same operacje.

Najpierw narzędzie Docker Machine sprawdza podstawowe elementy środowiska pracy, takie jak np. obecność zainstalowanego pakietu VirtualBox:

```
Running pre-create checks...
```

Po przeprowadzeniu sprawdzenia Docker Machine tworzy maszynę wirtualną (korzysta przy tym z wybranego sterownika):

```
Creating machine...
(docker-local) Copying /Users/russ/.docker/machine/cache/boot2docker.iso to
/Users/russ/.docker/machine/machines/docker-local/boot2docker.iso...
(docker-local) Creating VirtualBox VM...
(docker-local) Creating SSH key...
(docker-local) Starting the VM...
(docker-local) Check network to re-create if needed...
(docker-local) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
```

Jak widzisz, Docker Machine nadaje unikalny klucz SSH maszynie wirtualnej, co pozwala na uzyskanie dostępu do tej maszyny za pośrednictwem protokołu SSH (więcej informacji na ten temat znajdziesz w dalszej części książki). Po uruchomieniu maszyny wirtualnej Docker Machine łączy się z nią:

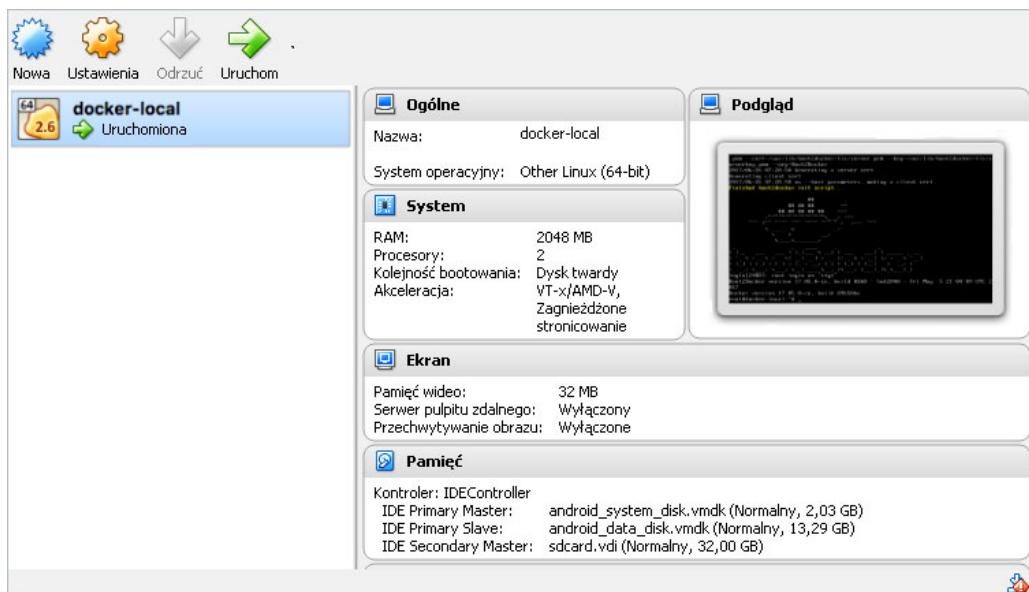
```
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
```

Jak widzisz, Docker Machine wykrywa system operacyjny i wybiera odpowiednie skrypty wdrażające Dockera. Po zainstalowaniu Dockera Docker Machine generuje i udostępnia certyfikaty (są one współdzielone przez lokalnego hosta i hosta Dockera). Następnie przeprowadzana jest konfiguracja zdalnej instalacji Dockera w celu wygenerowania uwierzytelniającego certyfikatu, dzięki któremu możliwe będzie połączenie się ze zdalnym serwerem Dockera z poziomu lokalnego klienta.

Na koniec Docker Machine sprawdza, czy lokalny klient Dockera może wykonać zdalne połączenie, a następnie kończy pracę, wyświetlając komunikat informujący o tym, jak należy skonfigurować lokalnego klienta w celu nawiązania połączenia z uruchomionym przed chwilą hostem Dockera:

```
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this
virtual machine, run: docker-machine env docker-local
```

Po otwarciu programu VirtualBox zobaczymy parametry swojej nowej maszyny wirtualnej:



Teraz musimy tak skonfigurować lokalnego klienta Dockera, aby mógł się połączyć z uruchomionym przed chwilą hostem Dockera. Informacje o tym, jak to zrobić, znajdziesz w wyświetlonym wcześniej komunikacie. W moim przypadku należało uruchomić polecenie:

```
$ docker-machine env docker-local
```

W wyniku uruchomienia tego polecenia wyświetlono następujące komunikaty:

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/russ/.docker/machine/machines/docker-local"
export DOCKER_MACHINE_NAME="docker-local"
# Run this command to configure your shell:
# eval $(docker-machine env docker-local)
```

Jak widzisz, nadpisana została lokalna instalacja Dockera — adres IP, port i certyfikat używany do uwierzytelniania zostały przypisane do uruchomionego przed chwilą hosta Dockera. Na końcu wyświetlanego komunikatu znajduje się polecenie, które należy uruchomić w celu skonfigurowania sesji Terminala łączącej się z nowym hostem.

Zacznijmy od sprawdzenia bieżącej konfiguracji za pomocą polecenia `docker version`:

```
russ in ~
⚡ docker version
Client:
Version: 17.03.1-ce
API version: 1.27
Go version: go1.7.5
Git commit: c6d412e
Built: Tue Mar 28 00:40:02 2017
OS/Arch: darwin/amd64

Server:
Version: 17.03.1-ce
API version: 1.27 (minimum version 1.12)
Go version: go1.7.5
Git commit: c6d412e
Built: Fri Mar 24 00:00:50 2017
OS/Arch: linux/amd64
Experimental: true
russ in ~
⚡
```

Na zrzucie zaprezentowano dane Dockera zainstalowanego przeze mnie w systemie macOS. Po uruchomieniu poniższej komendy polecenie `docker version` powinno zwracać inne dane serwera:

```
$ eval $(docker-machine env docker-local)
```

Tym razem `docker version` wyświetli następujące informacje:

```
russ in ~
⚡ eval $(docker-machine env docker-local)
russ in ~
⚡ docker version
Client:
Version: 17.03.1-ce
API version: 1.27
Go version: go1.7.5
Git commit: c6d412e
Built: Tue Mar 28 00:40:02 2017
OS/Arch: darwin/amd64

Server:
Version: 17.05.0-ce
API version: 1.29 (minimum version 1.12)
Go version: go1.7.5
Git commit: 89658be
Built: Thu May 4 21:43:09 2017
OS/Arch: linux/amd64
Experimental: false
russ in ~
⚡
```

Jak widzisz, zmieniła się wersja Dockera, wersja interfejsu API, a także data utworzenia. Od teraz możliwe jest korzystanie z hosta Dockera tak samo, jakby był on zainstalowany lokalnie.

Zanim przejdziemy do zagadnień związanych z uruchamianiem hostów Dockera w chmurze, chciałbym przedstawić kilka podstawowych poleceń obsługiwanych przez Docker Machine.

Oto polecenie wyświetlające listę skonfigurowanych hostów Dockera:

```
$ docker-machine ls
```

Po uruchomieniu tego polecenia wyświetlona zostanie następująca lista:

| NAME | ACTIVE | DRIVER | STATE | URL | SWARM | DOCKER | ERRORS |
|--------------|--------|------------|---------|---------------------------|-------|-------------|--------|
| docker-local | * | virtualbox | Running | tcp://192.168.99.100:2376 | | v17.05.0-ce | |
| russ | in ~ | | | | | | |

Lista zawiera nazwę maszyny, informacje o użytym sterowniku, adresie URL węzła końcowego Dockera i wersji Dockera.

W kolumnie *ACTIVE* (aktywność) mojej listy znajduje się symbol * świadczący o skonfigurowaniu hosta i możliwości korzystania z niego. W celu znalezienia aktywnych maszyn należy uruchomić polecenie `docker-machine active`.

Oto polecenie umożliwiające nawiązanie połączenia z hostem Dockera za pomocą protokołu SSH:

```
$ docker-machine ssh docker-local
```

Oto informacje wyświetcone po uruchomieniu tego polecenia:

Boot2Docker version 17.05.0-ce, build HEAD : 5ed2840 - Fri May 5 21:04:09 UTC 2017
Docker version 17.05.0-ce, build 89658be
docker@docker-local:~\$

Takie połączenie przydaje się w przypadku konieczności zainstalowania dodatkowego oprogramowania lub przeprowadzenia konfiguracji poza maszyną Docker Machine. Ponadto pozwala ono na przeglądanie dzienników i innych plików. W celu określenia adresu IP hosta Dockera uruchom polecenie:

```
$ docker-machine ip docker-local
```

Z polecenia tego będziemy korzystali często w dalszej części tego rozdziału. Polecenie `stop` zatrzymuje host Dockera, polecenie `restart` uruchamia go ponownie, a polecenie `remove` go usuwa. Oto przykłady użycia tych poleceń (na razie nie korzystaj z nich, aktywny host przyda Ci się podczas lektury dalszej części tego rozdziału):

```
$ docker-machine stop docker-local
$ docker-machine start docker-local
$ docker-machine restart docker-local
$ docker-machine rm docker-local
```

Oto polecenia, które służą do wyświetlania szczegółowych informacji dotyczących hosta Dockera:

```
$ docker-machine inspect docker-local
$ docker-machine config docker-local
$ docker-machine status docker-local
$ docker-machine url docker-local
```

Poznałeś już podstawowe wiadomości, czas zająć się czymś bardziej skomplikowanym.

Uruchamianie hostów Dockera w chmurze

W tym podrozdziale zajmiemy się sterownikami dwóch publicznych chmur. Przypominamy, że narzędzie Docker Machine ujednolica sposób obsługi różnych chmur.

DigitalOcean

W celu uruchomienia hosta Dockera w chmurze DigitalOcean za pomocą narzędzia Docker Machine musisz uzyskać kod dostępowy do interfejsu programistycznego tej chmury. Nie będziemy tutaj opisywać procesu generowania tego klucza. Skorzystaj z instrukcji znajdujących się na stronie <https://www.digitalocean.com/help/api/>.

Uruchomienie hosta Dockera za pomocą indywidualnego kodu dostępowego do interfejsu programistycznego chmury wiąże się z ponoszeniem wydatków. Cennik usług chmury DigitalOcean znajdziesz na stronie <https://www.digitalocean.com/pricing/>. Nie udostępniaj nikomu swojego kodu dostępowego, ponieważ może on zostać użyty w celu uzyskania nieautoryzowanego dostępu do Twojego konta. Wszystkie kody dostępowe, z których korzystamy w tym rozdziale, zostały zmodyfikowane — nie umożliwiają uzyskania dostępu do usługi. Do polecen narzędzia Docker Machine będziemy dodawać wiele flag, a więc polecenia umieszczone w książce zostaną podzielone na kilka linii za pomocą znaku \. Dzięki temu będą one bardziej czytelne.

W celu uruchomienia hosta o nazwie docker-digitalocean musimy skorzystać z następującego polecenia:

```
$ docker-machine create \
--driver digitalocean \
--digitalocean-access-token
ba46b9f97d16edb5a1f145093b50d97e50665492e9101d909295fa8ec35f20a1 \
docker-digitalocean
```

Host Dockera jest zdalną maszyną, a więc uruchomienie jej, skonfigurowanie i umożliwienie uzyskania do niej dostępu zajmuje trochę czasu. Przyjrzyj się poniższym komunikatom. Narzędzie Docker Machine uruchamia hosta Dockera w nieco inny sposób:

```
Running pre-create checks...
Creating machine...
(docker-digitalocean) Creating SSH key...
(docker-digitalocean) Creating Digital Ocean droplet...
(docker-digitalocean) Waiting for IP address to be assigned to the
Droplet...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with ubuntu(systemd)...
Installing Docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this
virtual machine, run: docker-machine env docker-digitalocean
```

Host po uruchomieniu powinien być widoczny w panelu sterowania chmury DigitalOcean:

The screenshot shows the DigitalOcean control panel. At the top, there's a navigation bar with links for 'Droplets', 'Images', 'Networking', 'Monitoring', 'API', and 'Support'. On the right side of the header is a green 'Create Droplet' button and a user profile icon. Below the header, the word 'Droplets' is prominently displayed. To the right of 'Droplets' is a search bar labeled 'Search by Droplet name'. Underneath, there are two tabs: 'Droplets' (which is selected and underlined in blue) and 'Volumes'. A table lists the currently active droplet. The table has columns for 'Name', 'IP Address', 'Created', and 'Tags'. One row is visible, showing a blue droplet icon, the name 'docker-digitalocean', the IP address '159.203.133.96', the creation time 'Just now', and a 'More' link.

Zmień konfigurację lokalnego klienta w celu połączenia się ze zdalnym hostem — uruchom polecenie:

```
$ eval $(docker-machine env docker-digitalocean)
```

Informacje na temat hosta Dockera uzyskasz, korzystając z poleceń `docker version` i `docker-machine inspect digitalocean`.

Po uruchomieniu polecenia `docker-machine ssh docker-digitalocean` połączysz się z hostem za pomocą protokołu SSH. Jeżeli przyjrzyisz się informacjom wyświetlonym w poniższym oknie, a także informacjom wyświetlonym podczas pierwszego uruchomienia hosta Dockera, to zauważysz, że w obu przypadkach host Dockera uruchomiono w innym systemie operacyjnym:

```
1. root@docker-digitalocean: ~ (docker-machine)
⚡ docker-machine ssh docker-digitalocean
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-81-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
  http://www.ubuntu.com/business/services/cloud

5 packages can be updated.
0 updates are security updates.

root@docker-digitalocean:~#
```

Jak widzisz, nie musielibyśmy informować narzędzia Docker Machine o tym, który system operacyjny ma zostać użyty, nie musielibyśmy określać rozmiaru hosta Dockera ani nawet ręcznie go uruchamiać. Wynika to z dość dobrych opcji domyślnych każdego z dostępnych sterowników. Po dodaniu opcji domyślnych do polecenia `docker-machine create` wyglądałoby ono tak:

```
$ docker-machine create \
--driver digitalocean \
--digitalocean-access-token
ba46b9f97d16edb5a1f145093b50d97e50665492e9101d909295fa8ec35f20a1 \
--digitalocean-image ubuntu-16-04-x64 \
--digitalocean-region nyc3 \
--digitalocean-size 512mb \
--digitalocean-ipv6 false \
--digitalocean-private-networking false \
--digitalocean-backups false \
--digitalocean-ssh-user root \
--digitalocean-ssh-port 22 \
docker-digitalocean
```

Jak widzisz, podczas uruchamiania hosta Dockera możesz ręcznie określić rozmiar, region, system operacyjny, a nawet opcje sieciowe.

Amazon Web Services

Usługi **Amazon Web Services (AWS)** są bardzo rozbudowane i oferują wiele możliwości. Wyjaśnienie procesu tworzenia użytkownika, generowania zabezpieczeń, a także instalowania i konfigurowania narzędzi umożliwiających korzystanie z usług AWS z poziomu wiersza poleceń zajęłoby kilkanaście stron. W związku z tym założę, że proces konfiguracji **Amazon Web Services** przeprowadziłeś już samodzielnie. Jeżeli tego nie zrobileś, a chciałbyś wykonać operacje przedstawione w tej sekcji, to zajrzyj do następujących dokumentów:

- Zabezpieczenia usług AWS: <https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html>.
- Dokumentacja interfejsu usług AWS obsługiwaneego z poziomu wiersza poleceń: <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html>.

Przed uruchomieniem hosta Dockera w chmurze AWS musimy ustalić identyfikator **prywatej chmury VPC** (ang. *Virtual Private Cloud*) w regionie `us-east-1` — regionie, w którym hosty Dockera są domyślnie uruchamianie. W tym celu należy uruchomić polecenie:

```
$ aws ec2 --region us-east-1 describe-vpcs
```

W moim przypadku polecenie to zwróciło następującą tablicę JSON:

```
{  
    "Vpcs": [  
        {  
            "VpcId": "vpc-35c91750",  
            "InstanceTenancy": "default",  
            "State": "available",  
            "DhcpOptionsId": "dopt-b0bfaf2",  
            "CidrBlock": "172.30.0.0/16",  
            "IsDefault": false  
        }  
    ]  
}
```

Jak widzisz, z tablicy tej bez problemów można odczytać identyfikator VPC ID. Teraz możemy uruchomić następujące polecenie:

```
docker-machine create \  
    --driver amazonec2 \  
    --amazonec2-vpc-id vpc-35c91750 \  
    docker-aws
```

Polecenie to połączy się z interfejsem programistycznym chmury AWS i uruchomi hosta Dockera:

```
Running pre-create checks...  
Creating machine...  
(docker-aws) Launching instance...  
Waiting for machine to be running, this may take a few minutes...  
Detecting operating system of created instance...  
Waiting for SSH to be available...  
Detecting the provisioner...  
Provisioning with ubuntu(systemd)...  
Installing Docker...  
Copying certs to the local machine directory...  
Copying certs to the remote machine...  
Setting Docker configuration on the remote daemon...  
Checking connection to Docker...  
Docker is up and running!  
To see how to connect your Docker Client to the Docker Engine running on this  
virtual machine, run: docker-machine env docker-aws
```

Jak widzisz, uruchamianie zadań wykonywanych przez maszynę Dockera pracującą w chmurze AWS wygląda tak samo jak w przypadku chmury DigitalOcean. Status hosta Dockera możesz sprawdzić, korzystając z polecenia:

```
$ aws ec2 --region us-east-1 describe-instances
```

Informacje na temat hosta Dockera znajdziesz również w konsoli usług AWS:

| Name | Instance ID | Instance Type | Availability Zone | Instance State | Status Checks | Alarm Status | Public DNS (IPv4) |
|------------|---------------------|---------------|-------------------|----------------|---------------|--------------|-------------------|
| docker-aws | i-016b90b3474d86f5e | t2.micro | us-east-1a | running | Initializing | None | - |

Description

| | |
|--|---|
| Instance ID: i-016b90b3474d86f5e (docker-aws) | Public IP: 184.73.76.74 |
| Instance state: running | Public DNS (IPv4): - |
| Instance type: t2.micro | IPv4 Public IP: 184.73.76.74 |
| Elastic IPs: - | IPv6 IPs: - |
| Availability zone: us-east-1a | Private DNS: ip-172-30-0-149.ec2.internal |
| Security groups: docker-machine, view inbound rules | Private IPs: 172.30.0.149 |
| Scheduled events: No scheduled events | Secondary private IPs: - |
| AMI ID: ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-20161221 (ami-9dcfdb8a) | VPC ID: vpc-35c91750 |
| | Subnet ID: subnet-61fbe227 |

Docker Machine, podobnie jak chmura DigitalOcean, korzysta z zestawu ustawień domyślnych. Gdyby dodać je do polecenia `create`, to wyglądałyby one następująco:

```
docker-machine create \
--driver amazonec2 \
--amazonec2-ami AWS_AMI ami-5f709f34 \
--amazonec2-region us-east-1 \
--amazonec2-vpc-id vpc-35c91750 \
--amazonec2-zone a \
--amazonec2-instance-type t2.micro \
--amazonec2-device-name /dev/sda1 \
--amazonec2-root-size 16 \
--amazonec2-volume-type gp2 \
--amazonec2-ssh-user ubuntu \
docker-aws
```

Jak widzisz, możesz zdefiniować wiele opcji określających miejsce i rozmiar uruchamianego hosta Dockera. Zwróć uwagę na to, że nie musielibyśmy przekazywać uwierzytelnień usług AWS. Wynika to z tego, że Docker Machine pobiera je bezpośrednio z konfiguracji AWS CLI. Uwierzytelnienia można znaleźć w katalogu `~/.aws/credentials`.

Od teraz możliwe jest korzystanie z usług AWS za pomocą tych samych poleceń Docker Machine co w przypadku chmury DigitalOcean i lokalnych hostów Dockera.

Więcej o sieciowych możliwościach Dockera

Uruchomiliśmy trzy hosty Dockera, które znajdują się w różnych miejscach. Teraz możemy przyjrzeć się kolejnym zagadnieniom związanym z pracą w sieci. Zamiast korzystać z domyślnego sterownika sieci, skorzystamy z rozszerzenia **Weave Net**, które umożliwia pracę sieciową z wieloma hostami. Rozszerzenie to zostało opracowane przez niezależną firmę Weave (<https://www.weave.works>).

Dotychczas uruchamialiśmy kontenery, korzystając z sieci, w której znajdował się tylko jeden host. Docker oferuje również możliwość rozszerzania sieci — tworzenia sieci kontenerów obejmującej wiele hostów — a więc kontener, w którym uruchomiona jest aplikacja serwera sieciowego, może np. komunikować się z bazą danych uruchomioną w kontenerze znajdującym się w innym hoście Dockera, korzystając przy tym z tego samego hosta DNS i zachowując taki sam poziom izolacji jak w przypadku umieszczenia obu kontenerów w tym samym hoście Dockera.

Zacznijmy od sprawdzenia funkcjonowania naszych trzech hostów:

```
$ docker-machine ls
```

W oknie Terminala pojawi się lista hostów Dockera:

| NAME | ACTIVE | DRIVER | STATE | URL | SWARM | DOCKER | ERRORS |
|---------------------|--------|--------------|---------|---------------------------|-------|-------------|--------|
| docker-aws | - | amazonene2 | Running | tcp://184.73.76.74:2376 | | v17.05.0-ce | |
| docker-digitalocean | * | digitalocean | Running | tcp://159.203.133.96:2376 | | v17.05.0-ce | |
| russ | - | virtualbox | Running | tcp://192.168.99.100:2376 | | v17.05.0-ce | |

Jak widzisz, w mojej konfiguracji host docker-digitalocean jest obecnie aktywny. Nie musimy przejmować się tym, która maszyna jest aktywna, ponieważ w poleceniach docker-machine i docker container będziemy korzystać z ich nazw.

Zanim zaczniemy uruchamiać kontenery, musimy zainstalować i skonfigurować dodatek Weave na wszystkich trzech hostach Dockera. Chmura DigitalOcean jest publicznie dostępna i nie ma żadnych zewnętrznych zapór połączenia (zapory takie chronią chmurę AWS), a zatem w celu ułatwienia host DigitalOcean będzie naszym głównym serwerem.

Aby zainstalować i skonfigurować dodatek Weave na hoście docker-digitalocean, musimy uruchomić następujące polecenia:

```
$ docker-machine ssh docker-digitalocean 'sudo curl -L git.io/weave -o /usr/local/bin/weave; sudo chmod a+x /usr/local/bin/weave'
$ docker-machine ssh docker-digitalocean sudo weave launch --password Wdmv3DzAvqR8NMGZEmZrJDdh
```

Nie podałem tutaj mojego prawdziwego hasła, tylko losowy ciąg znaków. Pamiętaj o tym, że musisz korzystać z własnego kodu dostępowego do chmury. Po skonfigurowaniu hosta docker-digitalocean możemy przystąpić do konfiguracji dwóch pozostałych hostów. W ich przypadku musimy uruchomić o jedno polecenie więcej, co wyjaśnimy po zakończeniu tego procesu.

W przypadku hosta docker-aws należy uruchomić następujące polecenia:

```
$ docker-machine ssh docker-aws 'sudo curl -L git.io/weave -o
/usr/local/bin/weave; sudo chmod a+x /usr/local/bin/weave'
$ docker-machine ssh docker-aws sudo weave launch --password
Wdmv3DzAvqR8NMGZEmZrJDdh
$ docker-machine ssh docker-aws sudo weave connect "$(docker-machine ip
docker-digitalocean)"
```

Aby skonfigurować host docker-local, uruchom następujące polecenia:

```
$ docker-machine ssh docker-local 'sudo curl -L git.io/weave -o
/usr/local/bin/weave; sudo chmod a+x /usr/local/bin/weave'
$ docker-machine ssh docker-local sudo weave launch --password
Wdmv3DzAvqR8NMGZEmZrJDdh
$ docker-machine ssh docker-local sudo weave connect "$(docker-machine ip
docker-digitalocean)"
```

Jak widzisz, mogliśmy przekazać nazwę hosta Dockera, nie musielibyśmy określać adresu IP hosta docker-digitalocean — posłużyliśmy się poleceniem docker-machine ip w celu wygenerowania tego adresu w sposób dynamiczny.

Po poleceniu docker-machine ssh przekazywaliśmy instrukcje, które mają być wykonane na każdym z trzech hostów, a więc nigdy nie korzystamy z protokołu SSH w celu bezpośredniego połączenia się z hostem.

Status sieci Weave Net możemy sprawdzić w dowolnym momencie za pomocą polecenia:

```
$ docker-machine ssh docker-digitalocean sudo weave status
```

Ze zwróconych informacji wynika, że wszystkie hosty są podłączone do sieci i komunikują się ze sobą:

```
Version: 2.0.0 (up to date; next check at 2017/06/26 16:02:13)

      Service: router
      Protocol: weave 1..2
            Name: 7e:f1:8e:70:0c:87(docker-digitalocean)
Encryption: enabled
PeerDiscovery: enabled
    Targets: 0
Connections: 2 (2 established)
      Peers: 3 (with 4 established connections)
TrustedSubnets: none
```

```
Service: ipam
    Status: idle
        Range: 10.32.0.0/12
DefaultSubnet: 10.32.0.0/12

Service: dns
    Domain: weave.local.
    Upstream: 8.8.4.4, 8.8.8.8
        TTL: 1
    Entries: 0

Service: proxy
    Address: unix:///var/run/weave/weave.sock

Service: plugin (legacy)
    DriverName: weave
```

Poleceniem status możesz również sprawdzić stan hostów docker-aws i docker-local:

```
$ docker-machine ssh docker-aws sudo weave status
$ docker-machine ssh docker-local sudo weave status
```

Po skonfigurowaniu sieci obejmującej kilka hostów możemy przystąpić do uruchamiania kontenerów. Przyjrzyjmy się procesowi uruchamiania aplikacji moby-counter, z której korzystaliśmy w poprzednim rozdziale. Zaczniemy od uruchomienia kontenera bazy Redis:

```
docker $(docker-machine config docker-local) container run -d \
--name redis \
--hostname="redis.weave.local" \
--network weave \
--dns="172.17.0.1" \
--dns-search="weave.local" \
redis:alpine
```

Teraz możemy uruchomić kontener moby-counter:

```
docker $(docker-machine config docker-digitalocean) container run -d \
--name moby-counter \
--network weave \
--dns="172.17.0.1" \
--dns-search="weave.local" \
-p 80:80 \
russmckendrick/moby-counter
```

Teraz uruchomimy jeszcze jedną kopię tego kontenera:

```
docker $(docker-machine config docker-aws) container run -d \
--name moby-counter \
--network weave \
--dns="172.17.0.1" \
--dns-search="weave.local" \
-p 80:80 \
russmckendrick/moby-counter
```

W celu skonfigurowania kontenerów pod kątem korzystania z usług DNS wbudowanych w narzędzie Weave Net musielibyśmy dodać kilka flag do polecień uruchamiających kontenery. Ponadto musielibyśmy jawnie skonfigurować nazwę hosta kontenera serwera Redis. Dzięki temu usługa DNS jest poprawnie zarejestrowana w sieci Weave Net.

Dodatkowo zmieniliśmy polecenie `docker-machine config` w celu zmodyfikowania konfiguracji klienta Dockera, tak aby nasz klient łączył się z wybranym hostem „w locie” — bez konieczności otwierania wielu terminali i zmiany konfiguracji zmiennych środowiskowych kolejnych hostów.

Otwórzmy przeglądarkę i wejdźmy pod adres IP hostów DigitalOcean i AWS. Użytkownicy systemów macOS i Linux mogą to zrobić za pomocą następujących polecień:

```
$ open http://$(docker-machine ip docker-digitalocean)/
$ open http://$(docker-machine ip docker-aws)/
```

Powinniśmy móc umieścić ikony w jednym oknie przeglądarki i zobaczyć je w takim samym układzie po odświeżeniu zawartości drugiego okna.

Kontener Redis można poddać operacji pingowania za pomocą polecień:

```
$ docker $(docker-machine config docker-digitalocean) container exec mobycounter
  ↵ping -c 3 redis
```

Po ich uruchomieniu w oknie Terminala wyświetlane zostaną następujące komunikaty:

```
russ in ~
$ docker $(docker-machine config docker-digitalocean) container exec mobycounter ping -c 3 redis
PING redis (10.32.0.1): 56 data bytes
64 bytes from 10.32.0.1: seq=0 ttl=64 time=85.384 ms
64 bytes from 10.32.0.1: seq=1 ttl=64 time=85.089 ms
64 bytes from 10.32.0.1: seq=2 ttl=64 time=103.115 ms

--- redis ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 85.089/91.196/103.115 ms
russ in ~
```

Oto polecenie wykonujące operację pingowania kontenera aplikacji Moby Counter z kontenera Redis:

```
$ docker $(docker-machine config docker-aws) container exec moby-counter
  ↵ping -c 3 redis
```

Oto efekt jego uruchomienia:

```
russ in ~
$ docker $(docker-machine config docker-aws) container exec moby-counter ping -c 3 redis
PING redis (10.32.0.1): 56 data bytes
64 bytes from 10.32.0.1: seq=0 ttl=64 time=93.025 ms
64 bytes from 10.32.0.1: seq=1 ttl=64 time=392.355 ms
64 bytes from 10.32.0.1: seq=2 ttl=64 time=92.898 ms

--- redis ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 92.898/192.759/392.355 ms
russ in ~
```

Obie operacje zwracają ten sam adres (w moim przypadku było to IP 10.32.0.1).

Kontenery moby-counter umieszczone w dwóch różnych publicznych chmurach współpracują z kontenerem uruchomionym w maszynie wirtualnej VirtualBox lokalnego komputera! Czy to nie wspaniałe?

Jeżeli próbowałeś tworzyć takie sieci w przeszłości, to na pewno potwierdzisz, że możliwość uruchomienia sieci z wieloma hostami — sieci łączącej publiczne chmury ze środowiskiem lokalnej maszyny wirtualnej — przy użyciu zaledwie ośmiu poleceń jest imponująca.

Przed przejściem do kolejnego podrozdziału skasujmy trzy utworzone hosty:

```
$ docker-machine rm docker-local docker-digitalocean docker-aws
```

Na ekranie pojawi się ostrzeżenie:

```
About to remove docker-local, docker-digitalocean, docker-aws
WARNING: This action will delete both local reference and remote instance.
Are you sure? (y/n): y
Successfully removed docker-local
Successfully removed docker-digitalocean
Successfully removed docker-aws
```

Otwórz panel sterowania DigitalOcean i konsolę AWS w celu sprawdzenia poprawności zakończenia pracy hostów — dzięki temu unikniesz ponoszenia niespodziewanych opłat.

Stosowanie innych bazowych systemów operacyjnych

Nie musisz korzystać z domyślnego systemu operacyjnego Docker Machine. Maszyna ta obsługuje również inne bazowe systemy operacyjne (dotyczy to również systemów zoptymalizowanych z myślą o kontenerach). W tym podrozdziale przyjrzymy się uruchamianiu systemów RancherOs i CoreOS na dwóch różnych platformach.

Obie wymienione dystrybucje, podobnie jak domyślny bazowy system Dockera — MobyOS, pozwalają na uruchomienie jądra systemowego, stoso sieciowego i kontenerów.

System CoreOS w chmurze DigitalOcean

System CoreOS obsługuje własne środowisko wykonawcze RTK (czytaj jak ang. *Rocket* — rakieta), ale współpracuje również z Dockerem. Niestety wersja Dockera dołączana do stabilnej wersji systemu CoreOS jest nieco starsza, o czym za chwilę się przekonamy.

W celu uruchomienia kontenera coreos-stable w chmurze DigitalOcean skorzystaj z polecenia:

```
docker-machine create --driver=digitalocean \
--digitalocean-accessstoken=ba46b9f97d16edb5a1f145093b50d97e50665492e9101d90929
↳5fa8ec35f20a1 \
--digitalocean-image=coreos-stable \
--digitalocean-size=1GB \
--digitalocean-ssh-user=core \
docker-coreos
```

Komunikaty wyświetcone w oknie Terminala przypominają te, które pojawiły się w nim podczas uruchamiania hostów Dockera w publicznych chmurach. Zwróć uwagę na to, że Docker Machine korzysta z zabezpieczeń właściwych systemowi CoreOS. Teraz możesz uruchomić następujące polecenie:

```
$ docker-machine ssh docker-coreos cat /etc/*release
```

Na ekranie pojawi się zawartość pliku *release* zawierającego informacje o dystrybucji:

```
DISTRIB_ID="Container Linux by CoreOS"
DISTRIB_RELEASE=1409.5.0
DISTRIB_CODENAME="Ladybug"
DISTRIB_DESCRIPTION="Container Linux by CoreOS 1409.5.0 (Ladybug)"
NAME="Container Linux by CoreOS"
ID=coreos
VERSION=1409.5.0
VERSION_ID=1409.5.0
BUILD_ID=2017-06-22-2222
PRETTY_NAME="Container Linux by CoreOS 1409.5.0 (Ladybug)"
ANSI_COLOR="38;5;75"
HOME_URL="https://coreos.com/"
BUG_REPORT_URL="https://issues.coreos.com"
COREOS_BOARD="amd64-usr"
```

Uruchomienie poniższego polecenia wyświetli więcej informacji na temat wersji Dockera uruchomionej w hoście CoreOS:

```
$ docker $(docker-machine config docker-coreos) version
```

Oto informacje zwrócone przez to polecenie. Zauważ, że wersja ta nie jest aktualna:

```
Client:
Version: 17.03.1-ce
API version: 1.24 (downgraded from 1.27)
Go version: go1.7.5
Git commit: c6d412e
Built: Tue Mar 28 00:40:02 2017
OS/Arch: darwin/amd64
```

```
Server:
Version: 1.12.6
API version: 1.24 (minimum version )
Go version: go1.7.5
```

```
Git commit: a82d35e
Built: Mon Jun 19 23:04:34 2017
OS/Arch: linux/amd64
Experimental: false
```

W związku z tym nie wszystkie polecenia przedstawione w tej książce będą działały poprawnie w tej wersji Dockera. W celu usunięcia hosta CoreOS uruchom polecenie:

```
$ docker-machine rm docker-coreos
```

Więcej informacji na temat systemu CoreOS znajdziesz na stronie <https://coreos.com/os/docs/latest/>. Jeżeli interesuje Cię środowisko RKT, to zajrzyj na stronę <https://coreos.com/rkt/>.

System RancherOS w maszynie VirtualBox

Rozdział 9. „Rancher” jest w całości poświęcony projektowi Rancher, ale nie opisuje on w pełni tego systemu operacyjnego zoptymalizowanego pod kątem kontenerów. We wspomnianym rozdziale skupimy się na projekcie Rancher jako narzędziu orkiestracji. System RancherOS jest uzupełnieniem systemu orkiestracji — projektu Rancher (<https://rancher.com>).

W celu uruchomienia hosta RancherOS w lokalnej maszynie VirtualBox skorzystaj z polecenia:

```
docker-machine create -d virtualbox \
--virtualbox-boot2docker-url
https://releases.rancher.com/os/latest/rancheros.iso
docker-rancher
```

W oknie Terminala pojawi się następujący komunikat:

```
(docker-rancher) Boot2Docker URL was explicitly set to "https://releases.rancher.com/os/latest/rancheros.iso" at create time, so Docker Machine cannot
➥upgrade this machine to the latest version.
```

Informuje on o tym, że nie będziesz mógł korzystać z polecenia `docker-machine upgrade`. Docker Machine będzie korzystać z systemu zabezpieczeń RancherOS. Po zakończeniu uruchamiania hosta możesz sprawdzić informacje o dystrybucji:

```
$ docker-machine ssh docker-rancher cat /etc/*release
```

Po uruchomieniu tego polecenia na ekranie pojawią się dane podobne do:

```
DISTRIB_ID=RancherOS
DISTRIB_RELEASE=v1.0.2
DISTRIB_DESCRIPTION="RancherOS v1.0.2"
NAME="RancherOS"
VERSION=v1.0.2
ID=rancheros
ID_LIKE=
VERSION_ID=v1.0.2
PRETTY_NAME="RancherOS v1.0.2"
```

```
HOME_URL="http://rancher.com/rancher-os/"
SUPPORT_URL="https://forums.rancher.com/c/rancher-os"
BUG_REPORT_URL="https://github.com/rancher/os/issues"
BUILD_ID=
```

Sprawdźmy zainstalowaną wersję Dockera:

```
$ docker $(docker-machine config docker-rancher) version
```

Ze zwróconych danych wynika, że wersja Dockera dołączona do systemu RanchOS jest nowsza od wersji wchodzącej w skład systemu CoreOS:

```
Client:
  Version: 17.03.1-ce
  API version: 1.27
  Go version: go1.7.5
  Git commit: c6d412e
  Built: Tue Mar 28 00:40:02 2017
  OS/Arch: darwin/amd64
```

```
Server:
  Version: 17.03.1-ce
  API version: 1.27 (minimum version 1.12)
  Go version: go1.7.5
  Git commit: c6d412e
  Built: Tue Mar 28 00:40:02 2017
  OS/Arch: linux/amd64
  Experimental: false
```

W celu usunięcia lokalnego hosta RancherOS wystarczy uruchomić polecenie `docker-machine rm -f docker-rancher`.

Więcej informacji na temat systemu RancherOS znajdziesz na stronie projektu Rancher (<http://rancher.com/rancher-os/>) i w repozytorium GitHub (<https://github.com/rancher/os/>).

Podsumowanie

W tym rozdziale przyjrzaliśmy się zagadnieniom związanym z narzędziem Docker Machine. Zaczęliśmy od opisania zastosowania tego narzędzia w celu tworzenia lokalnych hostów Dockera w maszynie wirtualnej VirtualBox, a następnie przedstawiliśmy polecenia umożliwiające zarządzanie hostami Dockera uruchomionymi za pomocą narzędzia Docker Machine. W dalszej części rozdziału opisaliśmy proces wdrażania hostów Dockera w środowiskach chmur DigitalOcean i Amazon Web Services.

Uruchomiliśmy trzy hosty zarządzane za pomocą Docker Machine, a następnie utworzyliśmy sieć pozwalającą na komunikowanie się tych hostów. Zrobiliśmy to przy użyciu narzędzia Weave Net. Po skonfigurowaniu sieci uruchomiliśmy aplikację opartą na wielu kontenerach, ale tym razem kontenery znajdowały się w różnych hostach.

Na koniec skrótnie opisaliśmy uruchamianie dwóch różnych systemów operacyjnych Linux zoptymalizowanych z myślą o kontenerach — CoreOS i RancherOS — i przedstawiliśmy różnice pomiędzy nimi.

Z pewnością zgodzisz się, że maszyna Docker Machine bardzo ułatwia wykonywanie zadań opisanych w tym rozdziale. Bez niej wykonanie ich trwałoby dłużej i byłoby o wiele bardziej skomplikowane.

W kolejnym rozdziale przyjrzymy się narzędziu Docker Compose służącemu do uruchamiania aplikacji korzystających z wielu kontenerów. Jest to jeden z głównych komponentów ekosystemu Dockera, z którego korzysta się praktycznie codziennie.

Docker Compose

W tym rozdziale opiszemy zagadnienia związane z kolejnym ważnym narzędziem ekosystemu Dockera — Docker Compose. W kolejnych podrozdziałach zajmę się tematami:

- wprowadzenie do Docker Compose,
- własna aplikacja uruchamiana za pomocą Docker Compose,
- plik YAML,
- polecenia narzędzia Docker Compose.

Wprowadzenie do Docker Compose

W rozdziale 1. „Docker — wprowadzenie” wymieniliśmy problemy, które rozwiązuje Docker. Umożliwia on uruchamianie dwóch odizolowanych od siebie aplikacji na tym samym serwerze. Aplikacje te mogą w związku z tym korzystać z różnych wersji tego samego stosu oprogramowania — np. PHP 5.6 i PHP7.

W poprzednich rozdziałach zajmowaliśmy się uruchamianiem aplikacji korzystających z wielu kontenerów. Nie uruchamialiśmy oprogramowania w formie stosu.

Aplikacja `moby-counter` jest napisana w Node.js. Korzysta ona z zaplecza w formie bazy Redis. W związku z tym wymagała uruchomienia dwóch kontenerów: kontenera aplikacji i kontenera bazy.

Nie mieliśmy z tym żadnych problemów, bo aplikacja była dość prosta. Ręczne uruchamianie pojedynczych kontenerów ma swoje wady.

Gdybym na przykład chciał, aby mój kolega wdrożył tę samą aplikację, to musiałbym przekazać mu następujące polecenia:

```
$ docker image pull redis:alpine
$ docker image pull russmckendrick/moby-counter
$ docker network create moby-counter
$ docker container run -d --name redis --network moby-counter redis:alpine
$ docker container run -d --name moby-counter --network moby-counter -p 8080:80
→russmckendrick/moby-counter
```

No dobra, mógłbym pominąć dwa pierwsze polecenia, ponieważ obraz zostanie pobrany automatycznie podczas próby jego uruchomienia (o ile mój kolega nie pobralby go wcześniej), ale po uruchomieniu za pomocą aplikacji napotkamy pewne problemy. Będę musiał przekazywać koledze wszystkie instrukcje związane z pracą aplikacji. Z czasem może ich być naprawdę dużo.

Ponadto musiałbym jasno zaznaczyć, że instrukcje należy wykonywać po kolej. Z kolei w uwagach musiałbym opisać również wszelkie potencjalne problemy i ich rozwiązania.

Docker jest oprogramowaniem służącym do tworzenia obrazów i uruchamiania kontenerów, ale jego twórcy przewidzieli tego typu scenariusz. Dzięki Dockerowi nie musimy się przejmować niezgodnością środowisk, w których uruchamiane są aplikacje, ponieważ środowiska te mogą zostać dołączone do obrazów.

W związku z tym w lipcu 2014 roku Docker przejął małą brytyjską spółkę o nazwie **Orchard Laboratories**. Ich strona internetowa (<https://www.orchardup.com>) jest wciąż aktywna.

Orchard Laboratories miała w ofercie dwa produkty. Pierwszym była platforma hostingowa oparta na Dockerze (można ją traktować jako połączenie narzędzia Docker Machine i samego Dockera). Jej użytkownik za pomocą polecenia `orchid` mógł uruchomić maszynę hosta, a następnie wykonywać na niej polecenia. Przyjrzyj się następującemu przykładowemu ciągowi poleceń:

```
$ orchard hosts create
$ orchard docker run -p 6379:6379 -d orchardup/redis
```

Polecenia te uruchamiały hosta Dockera na platformie Orchard, a następnie aktywowały kontener Redis.

Drugim produktem Orchard Laboratories był otwarty projekt o nazwie **Fig**. Umożliwiał on określenie w pliku YAML struktury aplikacji działającej w wielu kontenerach. Plik YAML mógł być następnie użyty w celu zautomatyzowania procesu uruchamiania kontenerów. Zaletą tego rozwiązania było to, że plik YAML (`fig.yml`) mógł zostać z łatwością dołączony do innych plików Dockera wchodzących w skład bazy kodu.

Docker przejął Orchard Laboratories z powodu produktu o nazwie Fig. Po dość krótkim czasie od przejęcia usługa Orchard przestała być oferowana, a w lutym 2015 r. Fig stał się narzędziem Docker Compose.

Więcej informacji na temat przejęcia spółki Orchard Laboratories przez Dockera znajdziesz w następującym wpisie bloga: <https://blog.docker.com/2014/07/welcoming-the-orchard-and-fig-team/>.

W pierwszym rozdziale opisaliśmy proces instalacji Dockera w systemach macOS, Windows i Linux. Podczas instalacji Dockera instalowane jest również narzędzie Docker Compose, a więc nie będziemy zajmować się ponownie jego instalacją i od razu spróbujemy skorzystać z narzędzia Docker Compose w celu uruchomienia naszej aplikacji korzystającej z dwóch kontenerów.

Uruchamianie aplikacji za pomocą narzędzia Docker Compose

Przypominamy, że Docker Compose korzysta z pliku YAML. Zwykle nosi on nazwę *docker-compose.yml*. Zawiera on definicje opisujące uruchamianie aplikacji korzystającej z wielu kontenerów. Oto zawartość pliku YAML definiującego uruchamianie aplikacji korzystającej z dwóch kontenerów, którą opisaliśmy w rozdziale 4. „Zarządzanie kontenerami” i w rozdziale 5. „Docker Machine”:

```
version: "3"

services:
  redis:
    image: redis:alpine
    volumes:
      - redis_data:/data
    restart: always
  mobycounter:
    depends_on:
      - redis
    image: russmckendrick/moby-counter
    ports:
      - "8080:80"
    restart: always

volumes:
  redis_data:
```

Plik ten jest dość czytelny i nawet bez dogłębnej analizy każdej jego linii dość łatwo można określić wykonywane operacje. W celu uruchomienia aplikacji wystarczy przejść do folderu, w którym znajduje się plik *docker-compose.yml* i uruchomić polecenie:

```
$ docker-compose up
```

W oknie zostaną wyświetcone komunikaty świadczące o wykonaniu wielu operacji:

```

1. mobycounter (docker-compose)
russ in ~/Documents/Code/mastering-docker/chapter06/mobycounter on master*
⚡ docker-compose up
Creating network "mobycounter_default" with the default driver
Creating volume "mobycounter_redis_data" with default driver
Creating mobycounter_redis_1
Creating mobycounter_mobycounter_1
Attaching to mobycounter_redis_1, mobycounter_mobycounter_1
mobycounter_1 | using redis server
redis_1 | 1:C 27 Jun 01:32:04.699 # Warning: no config file specified, using the default config.
redis_1 | In order to specify a config file use redis-server /path/to/redis.conf
mobycounter_1 | -----
redis_1
mobycounter_1 | have host: redis
redis_1 | have port: 6379
redis_1
redis_1
redis_1
redis_1
redis_1
redis_1
redis_1
redis_1
redis_1 | 1:M 27 Jun 01:32:04.700 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
redis_1 | 1:M 27 Jun 01:32:04.700 # Server started, Redis version 3.2.9
redis_1 | 1:M 27 Jun 01:32:04.700 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
redis_1 | 1:M 27 Jun 01:32:04.700 * The server is now ready to accept connections on port 6379
mobycounter_1 | server listening on port: 80
mobycounter_1 | Connection made to the Redis server

```

Po przyjrzeniu się kilku pierwszym liniom wyświetlonych komunikatów widać, że narzędzie Docker Compose wykonało następujące operacje:

- Utworzono sieć o nazwie `mobycounter_default`, która została oparta na domyślnym sterowniku sieci. Nie prosiliśmy o to w pliku YAML. Wyjaśnimy to za chwilę.
- Utworzono wolumin o nazwie `mobycounter_redis_data` (ponownie użyto domyślnego sterownika). Wolumin ten ma być używany przez naszą aplikację.
- Uruchomiono dwa kontenery. Nadano im nazwy `mobycounter_redis_1` i `mobycounter_mobycounter_1`.

Zwrócić uwagę na przestrzeń nazw wielokontenerowej aplikacji — mechanizm Docker Compose dodał do wszystkiego prefiks `mobycounter`. Został on wygenerowany na podstawie nazwy folderu, w którym znajdował się plik Docker Compose.

Po uruchomieniu kontenerów `mobycounter_redis_1` i `mobycounter_mobycounter_1` Docker Compose podpiął je do sesji Terminala i wyświetlał generowane przez nie informacje. Z wyświetlonych danych wynika, że kontenery `redis_1` i `mobycounter_1` zostały uruchomione i zaczęły nawiązywać ze sobą interakcje.

Podeczas pracy z Docker Compose użycie polecenia `docker-compose up` spowoduje wykonywanie operacji w tle. Wciśnięcie kombinacji klawiszy *Ctrl+C* zatrzyma kontenery i przywróci dostęp do wcześniejszej sesji Terminala.

Plik YAML narzędzia Docker Compose

Zanim przejdziemy do kolejnych zagadnień związanych z narzędziem Docker Compose, zatrzymajmy się nad plikiem `docker-compose.yml`. Pliki takie tworzą serce narzędzia Docker Compose.

YAML jest akronimem słów *YAML Ain't Markup Language* (YAML nie jest językiem znaczników). Jest to uniwersalny język formalny używany do konfigurowania, a także definiowania danych w sposób czytelny z punktu widzenia człowieka. Wcięcia widoczne w zaprezentowanym wcześniej przykładzie są bardzo ważne — pomagają one w definiowaniu struktury danych. Więcej informacji na temat tego języka znajdziesz na stronie <http://www.yaml.org>.

Plik YAML aplikacji Mobycounter

Plik `docker-compose.yml` użyty w celu uruchomienia wielokontenerowej aplikacji jest po-dzielony na trzy sekcje.

Pierwsza sekcja określa użytą wersję języka definicji Docker Compose. Korzystamy z aktualnej wersji Dockera i narzędzia Docker Compose, a więc korzystamy z języka w wersji 3:

```
version: "3"
```

W kolejnej sekcji definiowane są nasze kontenery. Sekcja ta rozpoczyna się od nagłówka `services` (usługi). Charakteryzuje się ona następującym formatem:

```
services:  
--> nazwa kontenera:  
----> opcje kontenera  
--> nazwa kontenera:  
----> opcje kontenera
```

W naszym przykładzie zdefiniowaliśmy dwa kontenery. W celu zwiększenia czytelności kodu oddzieliłem od siebie definicje obu kontenerów:

```
services:  
  redis:  
    image: redis:alpine  
    volumes:  
      - redis_data:/data  
    restart: always  
  
  mobycounter:  
    depends_on:  
      - redis  
    image: russmckendrick/moby-counter  
    ports:  
      - "8080:80"  
    restart: always
```

Składnia użyta do definiowania usług jest podobna do składni używanej do uruchamiania kontenera za pomocą polecenia `docker container run`. Obie składnie są podobnie czytelne, ale gdyby przyjrzeć się im bliżej, to odkryłoby się, że pomiędzy nimi istnieje wiele różnic.

Polecenie `docker container run` nie obsługuje flag wykonujących między innymi podane niżej operacje:

- `image` — instrukcja informująca Docker Compose o tym, który obraz ma zostać pobrany i użyty. Instrukcja ta nie ma formy opcji dostępnej podczas uruchamiania polecenia `docker container run` z poziomu wiersza poleceń (możliwe jest uruchomienie tylko jednego kontenera). W poprzednich rozdziałach pisaliśmy o tym, że obraz jest zawsze definiowany pod koniec polecenia bez potrzeby przekazywania flagi.
- `volume` — odpowiednik flagi `--volume`, który może przyjmować wiele wolumenów. Umożliwia korzystanie tylko z wolumenów zdefiniowanych w pliku YAML (za chwilę przedstawimy więcej informacji na ten temat).
- `depends_on` — to polecenie nigdy nie zadziałałoby podczas wywołania polecenia `docker container run`, ponieważ odwołuje się tylko do jednego kontenera. W przypadku narzędzia Docker Compose polecenie `depends_on` jest używane w celu logicznego określenia kolejności uruchamianych kontenerów. Można je zastosować do uruchomienia kontenera B tylko wtedy, gdy kontener A został poprawnie uruchomiony.
- `ports` — to polecenie działa jak flaga `--publish` przyjmująca listę portów.

Jedną częścią sekcji `services`, która ma swój odpowiednik w formie flagi polecenia `docker container run`, jest:

- `restart` — instrukcja ta działa tak samo jak flaga `--restart` i przyjmuje identyczne jak ona argumenty.

W ostatniej sekcji pliku YAML narzędzia Docker Compose definiujemy wolumeny:

```
volume:  
  redis_data:
```

Aplikacja do głosowania

Zaprezentowany wcześniej plik YAML narzędzia Docker Compose to tylko prosty przykład. Czas przyjrzeć się bardziej skomplikowanemu plikowi tego typu — przeanalizujemy zagadnienia związane z budowaniem kontenerów i wielu sieci.

Na stronie <ftp://ftp.helion.pl/przyklady/dockaz.zip> znajdziesz archiwum z kodem aplikacji *example-voting-app* (przykładowa aplikacja do głosowania). Aplikacja ta została umieszczona w folderze *r06*. Jest to rozbudowana wersja przykładowej aplikacji dostępnej w oficjalnym repozytorium Dockera <https://github.com/docker-samples/>.

Aplikacja składa się z pięciu kontenerów, dwóch sieci i jednego wolumenu. Przeanalizujmy jej plik YAML:

```
version: "3"
```

```
services:
```

Nasz pierwszy kontener nazywa się *vote*. Zawiera aplikację napisaną w Pythonie i pozwalającą użytkownikom na wysyłanie głosów. Z poniższej definicji wynika, że nie korzystamy z gotowego obrazu bazowego, a budujemy go od zera (instrukcja *build*):

```
vote:
  build: ./vote
  command: python app.py
  volumes:
    - ./vote:/app
  ports:
    - "5000:80"
  networks:
    - front-tier
    - back-tier
```

Zaprezentowana w tym przykładzie instrukcja *build* informuje narzędzie Docker Compose o konieczności zbudowania kontenera na podstawie pliku Dockerfile znajdującego się w folderze *./vote*. Plik ten, jak na aplikację Pythona, jest dość prosty.

Po uruchomieniu kontenera jest w nim montowany folder *./vote* systemu plików hosta. Dochodzi do tego w wyniku przekazania ścieżki folderu, który chcemy zamontować, i określenia miejsca, w którym ten folder ma być zamontowany.

Wydajemy kontenerowi polecenie uruchomienia aplikacji *python app.py*. Mapujemy port 5000 hosta z portem 80 kontenera, a następnie podłączamy kontener do dwóch sieci o nazwach *front-tier* i *back-tier*.

Sieć *front-tier* będzie łączyć kontenery, których porty są mapowane do hosta, a sieć *back-tier* jest zarezerwowana dla kontenerów, których porty nie muszą być odsłaniane — pełni funkcję prywatnej, odizolowanej sieci.

Aplikacja vote wysyła głosy do kontenera redis:

```
redis:  
  image: redis:alpine  
  container_name: redis  
  ports: ["6379"]  
  networks:  
    - back-tier
```

Kontener ten korzysta z oficjalnego obrazu bazy Redis i nie jest budowany na podstawie pliku Dockerfile. Udostępniamy port 6379, ale robimy to tylko w sieci back-tier.

Ponadto określamy nazwę kontenera — używamy instrukcji `container_name` w celu nadania kontenerowi nazwy `redis`. Ma to na celu uniknięcie domyślnego nadania kontenerowi nazwy wygenerowanej przez Docker Compose.

Po zatwierdzeniu głosu i zapisaniu go w bazie Redis uruchamiany jest proces `worker`:

```
worker:  
  build:  
    context: ./worker  
  networks:  
    - back-tier
```

W kontenerze procesu `worker` uruchomiona jest aplikacja .NET, która ma za zadanie połączyć się z kontenerem `redis` i zarejestrować każdy głos poprzez przeniesienie go do bazy danych PostgreSQL działającej w kontenerze o nazwie `db`. Kontener ten jest budowany na podstawie pliku Dockerfile, ale tym razem korzystamy z instrukcji `context` i nie podajemy ścieżki folderu, w którym znajdują się plik Dockerfile i aplikacja. Instrukcja ta określa katalog roboczy, w którym uruchamiane jest polecenie `docker build`, i umożliwia definiowanie dodatkowych opcji, takich jak etykiety i zmiana nazwy pliku Dockerfile.

Kontener ten łączy kontenery `redis` i `db`. Nie wykonuje żadnych innych operacji. W związku z tym nie musi odsłaniać żadnych portów. Ponadto nie musi on komunikować się z żadnym z kontenerów połączonych siecią `front-tier`, a więc możemy go dodać tylko do sieci `back-tier`.

Kolejna sekcja kodu tworzy wspomniany wcześniej kontener `db`:

```
db:  
  image: postgres:9.4  
  container_name: db  
  volumes:  
    - "db-data:/var/lib/postgresql/data"  
  networks:  
    - back-tier
```

Korzystamy z oficjalnego obrazu (podobnie jak w przypadku kontenera `redis`). Zauważ, że w zaprezentowanym kodzie nie ma instrukcji odsłaniającej kod. Wynika to z tego, że port jest udostępniany jako domyślna opcja oficjalnego obrazu. Kontenerowi nadajemy określoną przez nas nazwę.

Kontener ten będzie przechowywał głosy, a więc tworzymy i montujemy wolumen pełniący funkcję miejsca trwałego zapisu bazy danych PostgreSQL.

Na koniec uruchamiamy kontener aplikacji Node.js, który łączy się z bazą danych PostgreSQL uruchomioną w kontenerze db i wyświetla wyniki głosowania w czasie rzeczywistym (podczas generowania głosów przez kontener vote):

```
result:
  build: ./result
  command: nodemon --debug server.js
  volumes:
    - ./result:/app
  ports:
    - "5001:80"
    - "5858:5858"
  networks:
    - front-tier
    - back-tier
```

Kontener jest budowany na podstawie pliku Dockerfile w sposób podobny do kontenera vote. Udostępniamy port 5001, z którym można się połączyć w celu podejrzenia wyników.

Ostatnia część pliku *docker-compose.yml* zawiera deklarację wolumenu bazy danych PostgreSQL i dwóch sieci łączących kontenery:

```
volumes:
  db-data:

networks:
  front-tier:
  back-tier:
```

Polecenie `docker-compose up` zwraca wiele informacji na temat tego, co dzieje się podczas uruchamiania aplikacji. Pierwszy rozruch aplikacji może trwać około 5 minut. Jeżeli nie wykonyujesz samodzielnie opisywanych przeze mnie operacji, to poniżej znajdziesz skrót tego, co się dzieje podczas uruchamiania.

Zaczynamy od utworzenia sieci i przygotowania wolumenów do użycia przez kontenery:

```
Creating network "examplevotingapp_front-tier" with the default driver
Creating network "examplevotingapp_back-tier" with the default driver
Creating volume "examplevotingapp_db-data" with default driver
```

Następnie budujemy obraz kontenera vote:

```
Building vote
Step 1/7 : FROM python:2.7-alpine
2.7-alpine: Pulling from library/python
90f4dba627d6: Pull complete
19bc0bb0be9f: Pull complete
61bef6da706e: Pull complete
```

```
3e41fdc0d6e2: Pull complete
Digest:
sha256:10be70ff30d71a7bb3e8e7cb42f4e76ad8c28696ebce8c2faf89c33a2521a0f6
Status: Downloaded newer image for python:2.7-alpine
    --> 8a72f6fe0cf4
Step 2/7 : WORKDIR /app
    --> f555eeac2fe1
Removing intermediate container 0ac950f7ec78
Step 3/7 : ADD requirements.txt /app/requirements.txt
    --> 2e86103004ae
Removing intermediate container 9c676447514c
Step 4/7 : RUN pip install -r requirements.txt
    --> Running in 644ea3ffe44e
[lots of python build output here]
    --> 46e89f5a6d00
Removing intermediate container 644ea3ffe44e
Step 5/7 : ADD . /app
    --> 562c96f91d0d
Removing intermediate container ff8d462cd77c
Step 6/7 : EXPOSE 80
    --> Running in f681463674c1
    --> c601a2812a00
Removing intermediate container f681463674c1
Step 7/7 : CMD gunicorn app:app -b 0.0.0.0:80 --log-file - --access-logfile
- --workers 4 --keep-alive 0
    --> Running in f0a6128dba6c
    --> bbebc8f099cd
Removing intermediate container f0a6128dba6c
Successfully built bbebc8f099cd
Successfully tagged examplevotingapp_vote:latest
WARNING: Image for service vote was built because it did not already exist.
To rebuild this image you must use `docker-compose build` or `dockercompose up --build`.
```

Po zakończeniu budowy obrazu vote rozpoczyna się budowa kontenera worker:

```
Building worker
Step 1/5 : FROM microsoft/dotnet:1.1.1-sdk
1.1.1-sdk: Pulling from microsoft/dotnet
10a267c67f42: Pull complete
fb5937da9414: Pull complete
9021b2326a1e: Pull complete
c63131473568: Pull complete
a4274048307f: Pull complete
61820b027a34: Pull complete
Digest:
sha256:7badca85a6b29361f7cf2a9b68cbaa111f215943bfe6d87910a1943db8a2a02f
Status: Downloaded newer image for microsoft/dotnet:1.1.1-sdk
    --> a97efbca0c48
Step 2/5 : WORKDIR /code
    --> 16894fefdf6b9
Removing intermediate container 90ccc84a396b
Step 3/5 : ADD src/Worker /code/src/Worker
```

```

--> 826032994894
Removing intermediate container af0d371cc328
Step 4/5 : RUN dotnet restore -v minimal src/Worker && dotnet publish -c
Release -o "./" "src/Worker/"
--> Running in 21db89055f71
Restoring packages for /code/src/Worker/Worker.csproj...
[lots of .net build output here]
Restore completed in 8.16 sec for /code/src/Worker/Worker.csproj.
NuGet Config files used:
/root/.nuget/NuGet/NuGet.Config
Feeds used:
https://api.nuget.org/v3/index.json
Installed:
  84 package(s) to /code/src/Worker/Worker.csproj
Microsoft (R) Build Engine version 15.1.548.43366
Copyright (C) Microsoft Corporation. All rights reserved.
Worker -> /code/src/Worker/bin/Release/netcoreapp1.0/Worker.dll
--> 0654c936945a
Removing intermediate container 21db89055f71
Step 5/5 : CMD dotnet src/Worker/Worker.dll
--> Running in 40068deb9dc3
--> 10fd31632c6c
Removing intermediate container 40068deb9dc3
Successfully built 10fd31632c6c
Successfully tagged examplevotingapp_worker:latest
WARNING: Image for service worker was built because it did not already exist. To
rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
```

Następnie pobierany jest obraz redis:

```

Pulling redis (redis:alpine)...
alpine: Pulling from library/redis
88286f41530e: Pull complete
07b1ac6c7a50: Pull complete
91e2e140ea27: Pull complete
08957ceaa2b3: Pull complete
acd3d12a6a79: Pull complete
4ad88df09080: Pull complete
Digest:
sha256:2b1600c032e7653d079e9bd1eb80df5c99733795691f5ae9bca451bec325b7ea
Status: Downloaded newer image for redis:alpine
```

Następnym pobranym obrazem będzie baza danych PostgreSQL — obraz kontenera db:

```

Pulling db (postgres:9.4)...
9.4: Pulling from library/postgres
9f0706ba7422: Pull complete
df3070b9fd62: Pull complete
945954562465: Pull complete
820e17b80256: Pull complete
4f9e8f8bc763: Pull complete
Digest:
sha256:4f831cae0dff7c1cbb8b35e3e31f7d0da622bee4af1735993c66bfefec1e618
Status: Downloaded newer image for postgres:9.4
```

Teraz pora na coś większego — budowę obrazu result. Kod Node.js jest dość rozwlekły, a więc podczas wykonywania sekcji npm pliku Dockerfile na ekranie wyświetlonych zostanie wiele komunikatów:

```
Building result
Step 1/11 : FROM node:5.11.0-slim
5.11.0-slim: Pulling from library/node
8b87079b7a06: Pull complete
a3ed95caeb02: Pull complete
1bb8eaf3d643: Pull complete
5674f5dccbc4: Pull complete
96a79bcf8a3b: Pull complete
Digest:
sha256:fb4e332730514c393a78a3f0be6b6e1a7f7f3a63c9e670d6ccb0d54d3b9c4985
[lots and lots of nodejs output]
npm WARN optional Skipping failed optional dependency
/nodemon/chokidar/fsevents:
npm WARN notsup Not compatible with your operating system or architecture:
fsevents@1.1.2
npm info ok
    ---> bad5c376013a
Removing intermediate container dc0060b5cc1d
Step 4/11 : ADD package.json /app/package.json
    ---> 8b1b09e5e073
Removing intermediate container 92a2233c2608
Step 5/11 : RUN npm config set registry http://registry.npmjs.org
    ---> Running in 5dfffa4d8fa69
npm info it worked if it ends with ok
npm info using npm@3.8.6
npm info using node@v5.11.0
npm info config set "registry" "http://registry.npmjs.org"
npm info ok
    ---> 40860dc8d467
Removing intermediate container 5dfffa4d8fa69
Step 6/11 : RUN npm install && npm ls
    ---> Running in 1f57b2914af3
[lots more nodejs output]
npm info ok
    ---> bdbe2d653e45
Removing intermediate container 1f57b2914af3
Step 7/11 : RUN mv /app/node_modules /node_modules
    ---> Running in 7957e5f08c8a
    ---> d309bf17c968
Removing intermediate container 7957e5f08c8a
Step 8/11 : ADD . /app
    ---> 507a74fd7ff6
Removing intermediate container e5c5227af370
Step 9/11 : ENV PORT 80
    ---> Running in 959913cc7838
    ---> f8eb2f5eec79
Removing intermediate container 959913cc7838
Step 10/11 : EXPOSE 80
```

```

---> Running in 2371b833bcd
---> 3f7228a42540
Removing intermediate container 2371b833bcd
Step 11/11 : CMD node server.js
---> Running in ce472d3bde87
---> 4117d314ff4a
Removing intermediate container ce472d3bde87
Successfully built 4117d314ff4a
Successfully tagged examplevotingapp_result:latest
WARNING: Image for service result was built because it did not already exist. To
rebuild this image you must use `docker-compose build` or `docker-compose up --build`.

```

Po pobraniu obrazów i zbudowaniu kontenerów Docker Compose może uruchomić naszą aplikację:

```

Creating examplevotingapp_worker_1 ... done
Creating examplevotingapp_vote_1 ... done
Creating redis ... done
Creating db ... done
Creating examplevotingapp_result_1 ... done
Attaching to redis, db, examplevotingapp_worker_1,
examplevotingapp_result_1, examplevotingapp_vote_1

```

Wyniki głosowania można znaleźć pod adresem <http://localhost:5001>. Nie oddano jeszcze żadnych głosów, a więc prezentowany jest domyślny wynik 50/50:



Głos na koty (*CATS*) lub psy (*DOGS*) można oddać za pomocą formularza dostępnego na stronie <http://localhost:5000/>:

Cats vs Dogs!

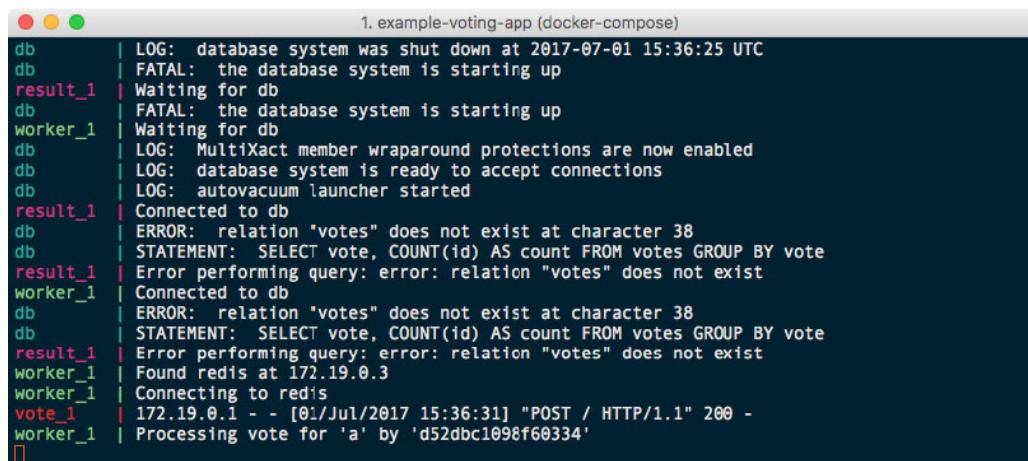
CATS

DOGS

(Tip: you can change your vote)

Processed by container ID
42b8bbcb39c8

Kliknięcie przycisku *CATS* lub *DOGS* spowoduje zarejestrowanie głosu. Informacje o tym zostaną umieszczone w dzienniku narzędzia Docker Compose wyświetlany w oknie Terminala:



```
1. example-voting-app (docker-compose)
db      | LOG:  database system was shut down at 2017-07-01 15:36:25 UTC
db      | FATAL: the database system is starting up
result_1 | Waiting for db
db      | FATAL: the database system is starting up
worker_1 | Waiting for db
db      | LOG: MultiXact member wraparound protections are now enabled
db      | LOG: database system is ready to accept connections
db      | LOG: autovacuum launcher started
result_1 | Connected to db
db      | ERROR: relation "votes" does not exist at character 38
db      | STATEMENT: SELECT vote, COUNT(id) AS count FROM votes GROUP BY vote
result_1 | Error performing query: error: relation "votes" does not exist
worker_1 | Connected to db
db      | ERROR: relation "votes" does not exist at character 38
db      | STATEMENT: SELECT vote, COUNT(id) AS count FROM votes GROUP BY vote
result_1 | Error performing query: error: relation "votes" does not exist
worker_1 | Found redis at 172.19.0.3
worker_1 | Connecting to redis
vote_1  | 172.19.0.1 - - [01/Jul/2017 15:36:31] "POST / HTTP/1.1" 200 -
worker_1 | Processing vote for 'a' by 'd52dbc1098f60334'
[]
```

W dzienniku znajduje się kilka komunikatów błędów wynikających z tego, że struktura tabeli *redis* jest tworzona dopiero po zarejestrowaniu pierwszego głosu przez aplikację *vote*. Po oddaniu głosu utworzona zostanie struktura tabeli *redis*, a kontener *worker* przetworzy ten głos i zapisze go w kontenerze *db*. Kontener wyników zostanie automatycznie zaktualizowany po oddaniu głosu:



W kolejnym rozdziale wróćmy do zagadnień związanych z plikami YAML narzędzia Docker Compose — zrobimy to przy okazji opisu uruchamiania stosu Docker Swarm. Na razie wróćmy do narzędzia Docker Compose i przeanalizujmy obsługiwane przez nie polecenia.

Polecenia Docker Compose

To już połowa rozdziału, a skorzystaliśmy tylko z jednego polecenia Docker Compose: `docker-compose up`. Jeżeli wykonywałeś opisane przez nas operacje, to po uruchomieniu polecenia `docker container ls -a` powinieneś zobaczyć następującą listę kontenerów:

```
$ ls in ~/Documents/Code/mastering-docker/chapter06/mobycounter on master*
$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
8fd4814abf9d      russmckendrick/moby-counter   "node index.js"    About a minute ago   Exited (137) 49 seconds ago
4ff901868303      redis:alpine          "docker-entrypoint..."  About a minute ago   Exited (0) 46 seconds ago
545776ba108       redis:alpine          "docker-entrypoint..."  About a minute ago   Up About an hour           0.0.0.0:32769->6379/tcp
4298bc9c39c8      examplevotingapp_vote     "python app.py"    About an hour ago   Up About an hour           0.0.0.0:5989->88/tcp
4c992be1e82c      examplevotingapp_worker_1  "/bin/sh -c dotnet ...  About an hour ago   Exited (1) 19 minutes ago
4d8095353055      examplevotingapp_result_1 "nodemon --debug ..."  About an hour ago   Up About an hour           0.0.0.0:5858->5858/tcp, 0.0.0.0:5001->80/tcp
9c4ee12c4ef4      postgres:9.4          "docker-entrypoint..."  About an hour ago   Up About an hour           5432/tcp
russ in ~/Documents/Code/mastering-docker/chapter06/mobycounter on master*
```

Wybierz jedną z aplikacji Docker Compose i przejdź do folderu, w którym znajduje się plik `docker-compose.yml`. Dzięki temu będziesz mógł sprawdzać działanie wybranych poleceń Docker Compose.

Up i PS

Zaczniemy od polecenia `docker-compose up`, ale tym razem dodamy do niego flagę. W folderze wybranej aplikacji uruchom polecenie:

```
$ docker-compose up -d
```

Spowoduje ono ponowne uruchomienie aplikacji, ale tym razem będzie ona działała w trybie odłączonym:

```
$ ls in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
$ docker-compose up -d
Starting examplevotingapp_worker_1 ...
examplevotingapp_result_1 is up-to-date
redis is up-to-date
Starting examplevotingapp_worker_1
examplevotingapp_vote_1 is up-to-date
Starting examplevotingapp_worker_1 ... done
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
```

Po przywróceniu kontroli nad Terminaliem możliwe jest sprawdzenie działania kontenerów za pomocą polecenia:

```
$ docker-compose ps
```

Ze zwróconych danych wynika, że wszystkie kontenery znajdują się w stanie aktywnym (Up):

| Name | Command | State | Ports |
|-------------------------------|--------------------------------|-------|-------------------------|
| db | docker-entrypoint.sh | Up | 5432/tcp |
| examplevotingapp_resu lt_1 | postgres | Up | 0.0.0.0:5858->5858/tcp |
| examplevotingapp_vote_1 | nodemon --debug server.js | Up | 0.0.0.0:5001->80/tcp |
| examplevotingapp_work er_1 | python app.py | Up | 0.0.0.0:5000->80/tcp |
| redis | /bin/sh -c dotnet src/Work ... | Up | 0.0.0.0:32769->6379/tcp |
| | redis ... | Up | redis ... |

```
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
```

Podeczas korzystania z tych poleceń narzędzie Docker Compose będzie świadome istnienia tylko tych kontenerów, które zostały zdefiniowane w sekcji service pliku *docker-compose.yml*. Wszystkie pozostałe kontenery będą ignorowane, ponieważ nie należą do stosu usług.

Config

Poniższe polecenie służy do sprawdzania pliku *docker-compose.yml*:

```
$ docker-compose config
```

W przypadku nieznalezienia żadnych błędów kopia pliku YAML zostanie wyświetlona na ekranie. Kopia ta przedstawia sposób interpretacji pliku przez narzędzie Docker Compose. Jeżeli nie chcesz oglądać tej reprezentacji pliku YAML i chcesz tylko sprawdzić, czy w pliku tym nie występują błędy, to skorzystaj z polecenia:

```
$ docker-compose config -q
```

Jest to skrót flagi `--quiet` (cicho). W zaprezentowanych przykładach nie występują błędy, ale w przypadku ich wykrycia zostaniesz o nich poinformowany poprzez następujące komunikaty:

```
ERROR: yaml.parser.ParserError: while parsing a block mapping
  in "./docker-compose.yml", line 1, column 1
    expected <block end>, but found '<block mapping start>'
  in "./docker-compose.yml", line 27, column 3
```

Pull, build i create

Kolejne dwa polecenia pomagają w przygotowaniu aplikacji Docker Compose do uruchomienia. Poniższe polecenie odczytuje zawartość pliku YAML i pobiera wszystkie niezbędne obrazy:

```
$ docker-compose pull
```

Oto polecenie wykonujące wszystkie instrukcje `build` znajdujące się w pliku YAML:

```
$ docker-compose build
```

Te polecenia przydają się do testowania definiowanej aplikacji bez uruchamiania jej. Polecenie `docker-compose build` może być również użyte w celu aktywacji procesu budowy obrazów po zmodyfikowaniu któregoś z plików Dockerfile użytego podczas wcześniejszego procesu budowy obrazów.

Polecenia `pull` i `build` pobierają i budują obrazy aplikacji — nie przeprowadzają procesu konfiguracji kontenerów. W celu wykonania tej operacji należy skorzystać z polecenia:

```
$ docker-compose create
```

Polecenie to utworzy wszystkie kontenery bez uruchamiania ich — wszystkie kontenery będą miały stan `Exited` aż do chwili, gdy je uruchomisz. Polecenie `create` obsługuje kilka przydatnych flag:

- `--force-recreate` — tworzy ponownie kontener nawet wtedy, gdy nie jest to konieczne (w przypadku braku zmian konfiguracji).
- `--no-recreate` — nie odtwarza kontenera, jeżeli on już istnieje (flagi tej nie można łączyć z flagą `--force-recreate`).
- `--no-build` — obraz nie jest budowany nawet wtedy, gdy go brakuje.
- `--build` — buduje obrazy przed utworzeniem kontenerów.

Start, stop, restart, pause i unpause

Wymienione polecenia działają tak samo jak ich odpowiedniki współpracujące z instrukcją `docker container`. Jedyną różnicą jest to, że poniższe polecenia wpływają na wszystkie kontenery:

```
$ docker-compose start
$ docker-compose stop
$ docker-compose restart
$ docker-compose pause
$ docker-compose unpause
```

Istnieje możliwość wpływania na pracę tylko jednej usługi. W tym celu należy przekazać jej nazwę. Oto przykład instrukcji wstrzymujących i wznowiających usługę `db`:

```
$ docker-compose pause db
$ docker-compose unpause db
```

Top, logs i events

Trzy polecenia opisane w tej sekcji dostarczają informacji na temat tego, co dzieje się w uruchomionych kontenerach i aplikacji Docker Compose.

Poniższe polecenie, podobnie jak jego odpowiednik współpracujący z instrukcją `docker container`, wyświetla informacje o procesach uruchomionych w każdym z kontenerów uruchomionych za pomocą Docker Compose:

```
$ docker-compose top
```

Zwrócone informacje są dzielone na sekcje zawierające dane poszczególnych kontenerów:

```
1. example-voting-app (bash)
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
⚡ docker-compose top
db
PID  USER   TIME           COMMAND
-----
3135  999   0:00  postgres
3472  999   0:00  postgres: checkpointer process
3473  999   0:00  postgres: writer process
3474  999   0:00  postgres: wal writer process
3475  999   0:00  postgres: autovacuum launcher process
3476  999   0:00  postgres: stats collector process
3480  999   3:50  postgres: postgres postgres 172.19.0.2(60348) idle
3517  999   0:00  postgres: postgres postgres 172.19.0.3(50578) idle

examplevotingapp_result_1
PID  USER   TIME           COMMAND
-----
3212  root   0:00  node /usr/local/bin/nodemon --debug server.js
3506  root   0:00  sh -c node --debug server.js
3507  root   0:01  node --debug server.js

examplevotingapp_vote_1
PID  USER   TIME           COMMAND
-----
3400  root   0:00  python app.py
3496  root   0:02  /usr/local/bin/python app.py

examplevotingapp_worker_1
PID  USER   TIME           COMMAND
-----
2994  root   0:00  /bin/sh -c dotnet src/Worker/Worker.dll
3099  root   16:14  dotnet src/Worker/Worker.dll

redis
PID  USER   TIME           COMMAND
-----
3311  chrony  1:46  redis-server
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
⚡
```

Jeżeli interesują Cię informacje na temat tylko jednej z usług, to musisz przekazać jej nazwę do polecenia top:

```
$ docker-compose top db
```

Kolejne polecenie wyświetla na ekranie dane z dzienników aktywnych kontenerów:

```
$ docker-compose logs
```

Podobnie jak w przypadku polecenia docker container, możesz korzystać z flag -f lub -follow w celu śledzenia informacji aż do wciśnięcia kombinacji klawiszy *Ctrl+C*. Po umieszczeniu nazwy wybranej usługi na końcu polecenia możesz przeglądać dziennik tylko tej aplikacji.

```
1. example-voting-app (docker-compose)
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
⚡ docker-compose logs -f db
Attaching to db
db    | LOG:  database system was shut down at 2017-07-01 18:42:04 UTC
db    | LOG:  MultiXact member wraparound protections are now enabled
db    | LOG:  database system is ready to accept connections
db    | LOG:  autovacuum launcher started
⚡
```

Polecenie events również działa tak jak wersja współpracująca z instrukcją docker container — strumieniuje ono w czasie rzeczywistym zdarzenia takie jak np. te, które zostały aktywowane przez inne opisane polecenia. Sprawdź działanie polecenia:

```
$ docker-compose events
```

Gdy teraz uruchomisz polecenie docker-compose pause, to w Terminalu pojawią się następujące komunikaty:

```
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
⚡ docker-compose events
2017-07-02 11:13:14.089008 container pause 826ff65e71a9f92c6de32898f794073d3d993d05d94ffd0e06a558225
b40ed63 (image=postgres:9.4, name=db)
2017-07-02 11:13:14.248001 container pause 8647676addf6068a61e807deca6cf4bf23198ffd2f25b2f69ec037cf
1899537 (image=redis:alpine, name=redis)
2017-07-02 11:13:14.248001 container pause c36650d033026dee86359436933f426dec999838f636fc9155f40f4f
1e50de1 (image=examplevotingapp_vote, name=examplevotingapp_vote_1)
2017-07-02 11:13:14.248186 container pause d4caa1b2c9af9334b228cf6fa3615d66680870f60d4fcbb0209068e8b9
484bc18 (image=examplevotingapp_result, name=examplevotingapp_result_1)
2017-07-02 11:13:14.248370 container pause 2d82adcad614aa11d297b4c891ebc5094e3917dd2225dd4e2c0f8ba97
c188c45 (image=examplevotingapp_worker, name=examplevotingapp_worker_1)
```

Exec i run

Polecenia exec i run działają podobnie do swych odpowiedników współpracujących z instrukcją docker container. Uruchom następujące przykładowe polecenie:

```
$ docker-compose exec worker ping -c 3 db
```

Uruchomi ono nowy proces w aktywnym kontenerze worker, a następnie wykona trzy próby pingowania kontenera db:

```
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
⚡ docker-compose exec worker ping -c 3 db
PING db (172.19.0.4): 56 data bytes
64 bytes from 172.19.0.4: icmp_seq=0 ttl=64 time=0.082 ms
64 bytes from 172.19.0.4: icmp_seq=1 ttl=64 time=0.037 ms
64 bytes from 172.19.0.4: icmp_seq=2 ttl=64 time=0.035 ms
--- db ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.035/0.051/0.082/0.000 ms
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
```

Polecenie run przydaje się do uruchamiania pojedynczych poleceń odwołujących się do kontenerów z poziomu aplikacji. Jeżeli korzystasz z menedżera pakietów, takiego jak np. composer, w celu zaktualizowania plików zależnych projektu przechowywanych w wolumenie, to możesz skorzystać z następującego polecenia:

```
$ docker-compose run --volume wolumen_danych:/app composer install
```

Polecenie to uruchamia kontener composer z poleceniem `install` i montuje wolumen `wolumen_danych` w folderze `/app` kontenera.

Scale

Polecenie `scale` przyjmuje przekazaną usługę i skaluje ją zgodnie ze zdefiniowaną wartością. Oto przykładowe polecenie, z którego mogę skorzystać, chcąc dodać więcej kontenerów `worker`:

```
$ docker-compose scale worker=3
```

Próba jego uruchomienia spowoduje wyświetlenie następującego ostrzeżenia:

```
WARNING: The scale command is deprecated. Use the up command with the --scale flag instead.
```

Wynika z niego, że powinniśmy tak naprawdę korzystać z następującego polecenia:

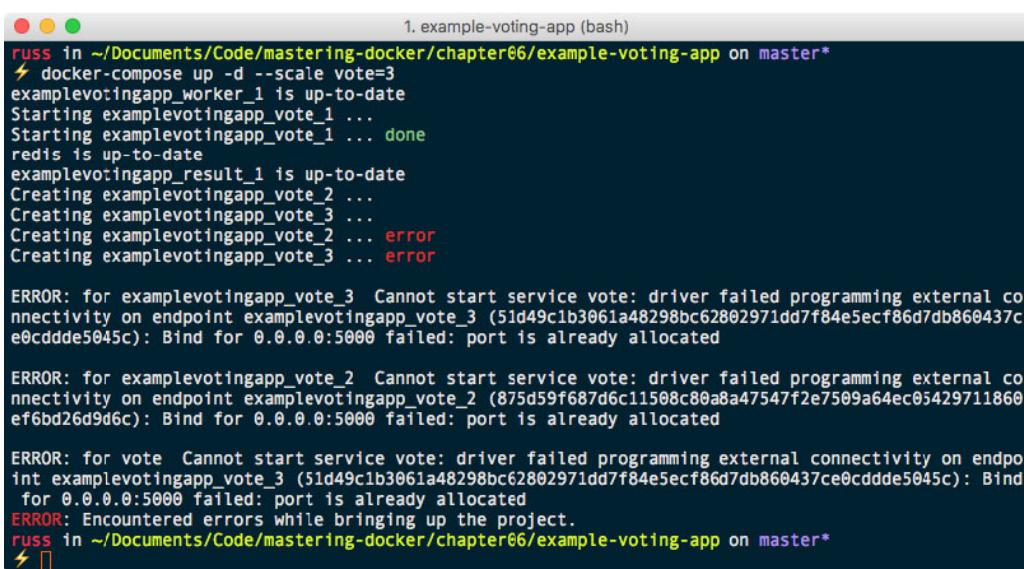
```
$ docker-compose up -d --scale worker=3
```

Aktualna wersja narzędzia Docker Compose obsługuje jeszcze polecenie `scale`, ale kolejne wersje Docker Compose nie będą już obsługiwały tego polecenia.

Zdecydowałem się na skalowanie liczby kontenerów `worker`. Mam ku temu dobry powód. Przekonaj się o tym samodzielnie, próbując uruchomić następujące polecenie:

```
$ docker-compose up -d --scale vote=3
```

Docker Compose utworzy dwa dodatkowe kontenery, ale próba ich uruchomienia zakończy się niepowodzeniem i wyświetleniem następującego komunikatu:



```
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app (bash)
$ docker-compose up -d --scale vote=3
examplevotingapp_worker_1 is up-to-date
Starting examplevotingapp_vote_1 ...
Starting examplevotingapp_vote_1 ... done
redis is up-to-date
examplevotingapp_result_1 is up-to-date
Creating examplevotingapp_vote_2 ...
Creating examplevotingapp_vote_3 ...
Creating examplevotingapp_vote_2 ... error
Creating examplevotingapp_vote_3 ... error

ERROR: for examplevotingapp_vote_3 Cannot start service vote: driver failed programming external connectivity on endpoint examplevotingapp_vote_3 (51d49c1b3061a48298bc62802971dd7f84e5ecf86d7db860437ce0cdde5045c): Bind for 0.0.0.0:5000 failed: port is already allocated

ERROR: for examplevotingapp_vote_2 Cannot start service vote: driver failed programming external connectivity on endpoint examplevotingapp_vote_2 (875d59f687d6c11508c80a8a47547f2e7509a64ec05429711860ef6bd26d9d6c): Bind for 0.0.0.0:5000 failed: port is already allocated

ERROR: for vote Cannot start service vote: driver failed programming external connectivity on endpoint examplevotingapp_vote_3 (51d49c1b3061a48298bc62802971dd7f84e5ecf86d7db860437ce0cdde5045c): Bind for 0.0.0.0:5000 failed: port is already allocated
ERROR: Encountered errors while bringing up the project.
russ in ~/Documents/Code/mastering-docker/chapter06/example-voting-app on master*
```

Wynika to z tego, że trzy kontenery nie mogą być mapowane do tego samego portu. Problem ten można rozwiązać za pomocą techniki przedstawionej w kolejnym rozdziale.

Kill, rm i down

W tej sekcji opiszymy trzy polecenia służące do usuwania i zamazywania aplikacji Docker Compose. Polecenie `kill` zatrzymuje natychmiast aktywne procesy kontenera:

```
$ docker-compose kill
```

Zachowaj ostrożność, korzystając z tego polecenia, ponieważ nie czeka ono na poprawne zatrzymanie pracy kontenera tak jak polecenie `docker-compose stop`, a więc korzystanie z polecenia `docker-compose kill` może wiązać się z ryzykiem utraty danych.

Polecenie `rm` usuwa wszystkie zatrzymane kontenery (kontenery ze statusem `exited`):

```
$ docker-compose rm
```

Polecenie `down` jest przeciwnieństwem polecenia `docker-compose up`:

```
$ docker-compose down
```

Usuwa one kontenery i sieci utworzone w wyniku uruchomienia polecenia `docker-compose up`. Jeżeli chcesz usunąć wszystko, to możesz to zrobić za pomocą polecenia:

```
$ docker-compose down --rmi all --volumes
```

Usunie one wszystkie kontenery, sieci, wolumeny, a także pobrane i zbudowane obrazy (wszystko to, co zostało wygenerowane w wyniku uruchomienia polecenia `docker-compose up`). Dotyczy to również obrazów, które mogą być używane poza aplikacją Docker Compose, ale próba usunięcia używanego obrazu zakończy się wyświetleniem komunikatu o błędzie:

```
russ in ~/Documents/Code/mastering-docker/chapter06/mobycounter on master*
⚡ docker-compose down --rmi all --volumes
Stopping mobycounter_mobycounter_1 ... done
Stopping mobycounter_redis_1 ... done
Removing mobycounter_mobycounter_1 ... done
Removing mobycounter_redis_1 ... done
Removing network mobycounter_default
Removing volume mobycounter_redis_data
Removing image redis:alpine
Removing image russmckendrick/moby-counter
russ in ~/Documents/Code/mastering-docker/chapter06/mobycounter on master*
⚡
```

Podsumowanie

Mam nadzieję, że spodobała Ci się lektura rozdziału o narzędziu Docker Compose i że zacząłeś tak jak ja uważać, iż ten niezależny projekt stał się bardzo ważną częścią ekosystemu Dockera.

Narzędzie Docker Compose wprowadza pewne koncepcje dotyczące uruchamiania kontenerów i zarządzania nimi. Koncepcje te zostaną rozwinięte w rozdziale 7. „Docker Swarm”.

Docker Swarm

W tym rozdziale zajmiemy się rozwiązyaniem Docker Swarm, które pozwala na tworzenie klastrów Dockera i zarządzanie nimi. Rozwiązywanie to może być używane do umieszczania kontenerów w różnych hostach, a także ich skalowania.

W tym rozdziale zajmiemy się następującymi zagadnieniami:

- instalacja narzędzia Docker Swarm,
- role Docker Swarm,
- zastosowania narzędzia Docker Swarm,
- usługi i stosy Docker Swarm,
- równoważenie obciążenia i tworzenie harmonogramów za pomocą narzędzia Docker Swarm.

Zanim przejdziemy do uruchamiania klastrów Docker Swarm, chciałbym poinformować Cię o tym, że obecnie dostępne są dwie wersje narzędzia Docker Swarm.

W pierwszym wydaniu tej książki rozdział ten dotyczył samodzielnej — obecnie uważanej za przestarzałą — wersji narzędzia Docker Swarm. Była ona obsługiwana do Dockera 1.12. Obecnie nie jest ona aktywnie rozwijana. Więcej informacji na jej temat znajdziesz w dokumentacji znajdującej się na stronie <https://docs.docker.com/swarm/>.

W momencie pisania tej książki (pierwszy kwartał 2017 r.) wersja 1.11.x Dockera nie zalecała już korzystania z samodzielnej wersji narzędzia Docker Swarm.

W Dockerze 1.12 wprowadzono tryb Docker Swarm. Tryb ten zapewniał obsługę wszystkich funkcji samodzielnego narzędzia Docker Swarm z poziomu jądra silnika Docker, a także wprowadzał dodatkowe funkcje. W tej książce piszemy o Dockerze w wersji 1.13 lub nowszej, a więc będziemy korzystać z trybu Docker Swarm. Od teraz, za każdym razem, gdy piszemy o Docker Swarm, mamy na myśli wbudowany w Dockera tryb pracy, a nie samodzielne narzędzie.

Docker Swarm — instalacja

Korzystamy z wersji Dockera, w której wbudowano obsługę trybu Docker Swarm, a więc nie musimy dodatkowo instalować narzędzia Docker Swarm. Dostępność trybu Docker Swarm możesz sprawdzić, uruchamiając polecenie:

```
$ docker swarm --help
```

W oknie Terminala powinny pojawić się informacje podobne do następujących:

```
russ in ~
⚡ docker swarm --help

Usage: docker swarm COMMAND

Manage Swarm

Options:
  --help    Print usage

Commands:
  init      Initialize a swarm
  join      Join a swarm as a node and/or manager
  join-token  Manage join tokens
  leave     Leave the swarm
  unlock    Unlock swarm
  unlock-key  Manage the unlock key
  update    Update the swarm

Run 'docker swarm COMMAND --help' for more information on a command.
russ in ~
⚡
```

Jeżeli zamiast nich zobaczyłeś komunikat błędu, to sprawdź, czy używasz Dockera w wersji 1.13 lub nowszej. W momencie pisania tej książki najnowsza wersja to 17.03.

Role Docker Swarm

Jakie role wiążą się z narzędziem Docker Swarm? Przyjrzymy się dwóm rołom, które może przyjąć host obsługujący klastry Docker Swarm:

- menedżer (*Swarm manager*),
- wykonawca (*Swarm worker*).

Menedżer Swarm

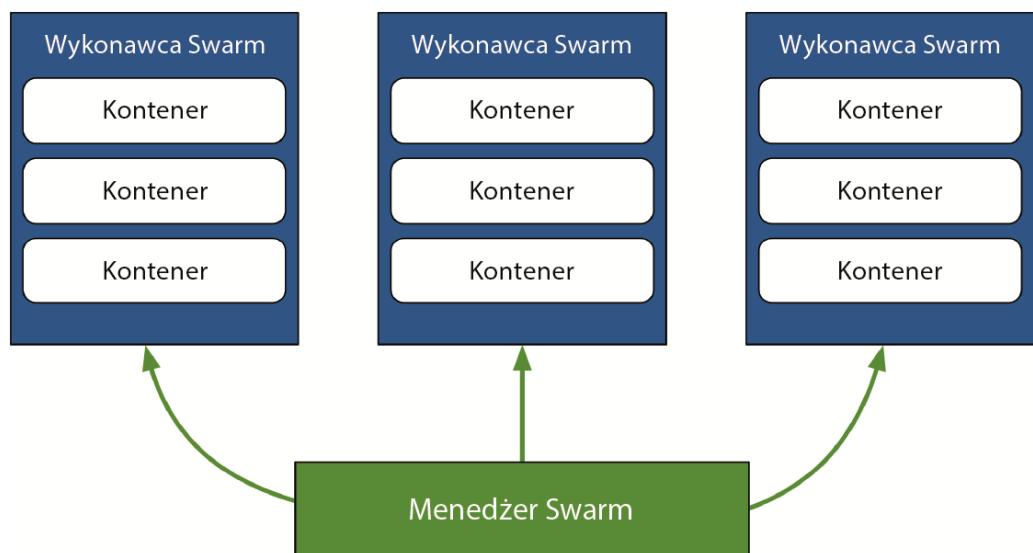
Host będący menedżerem jest centralnym zarządcą wszystkich hostów Swarm. Pozwala on na wydawanie poleceń sterujących pracą wszystkich węzłów. Możesz przełączać się pomiędzy poszczególnymi węzłami, łączyć je i usuwać, a także wykonywać różne operacje na hostach.

W każdym klastrze można uruchomić kilka menedżerów Swarm. W środowiskach produkcyjnych zaleca się uruchamianie przynajmniej pięciu menedżerów Swarm, co oznacza, że nasz kластer może mieć problemy z maksymalnie dwoma węzłami menedżera Swarm, zanim zacznie zwracać komunikaty błędów. Menedżerowie Swarm korzystają z algorytmu **Raft** (*Raft Consensus*) w celu utrzymania zgodnych stanów we wszystkich węzłach menedżera.

Jeżeli interesuje Cię dokładne wyjaśnienie algorytmów Raft, to polecam Ci lekturę doskonalej prezentacji opublikowanej w serwisie The Secret Lives of Data, znajdującej się pod adresem <http://thesecretlivesofdata.com/raft/>. Wyjaśnia ona procesy przeprowadzane w tle przez węzły menedżera.

Wykonawca Swarm

Wykonawcy Swarm to hosty Dockera, w których uruchomione są kontenery. Pracą wykonawców Swarm steruje **menedżer Swarm**.



Oto ilustracja zależności pomiędzy komponentami Docker Swarm. Wynika z niej, że **menedżer Swarm** komunikuje się z każdym z hostów, w których uruchomione są kontenery Swarm.

Korzystanie z trybu Docker Swarm

W tej sekcji przyjrzymy się korzystaniu z trybu Docker Swarm i wykonywaniu następujących zadań:

- tworzeniu klastra,
- łączeniu wykonawców,
- tworzeniu listy węzłów,
- zarządzaniu klastrem.

Tworzenie klastra

Zacznijmy od utworzenia klastra, który uruchamia się z menedżerem Swarm. Będziemy tworzyć wielowęzłowy klaster na komputerze lokalnym, a więc w celu uruchomienia hosta powinniśmy skorzystać z narzędzia Docker Machine:

```
$ docker-machine create \
-d virtualbox \
swarm-manager
```

Oto skrócona wersja komunikatów, które zostaną wyświetcone w oknie Terminala:

```
(swarm-manager) Creating VirtualBox VM...
(swarm-manager) Creating SSH key...
(swarm-manager) Starting the VM...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this
virtual machine, run: docker-machine env swarm-manager
```

Węzeł `swarm-manager` (menedżer Swarm) został właśnie uruchomiony w maszynie wirtualnej VirtualBox. Możemy to sprawdzić za pomocą polecenia:

```
$ docker-machine ls
```

Spowoduje ono wyświetlenie następującej listy:

| NAME | ACTIVE | DRIVER | STATE | URL | SWARM | DOCKER | ERRORS |
|---------------|--------|------------|---------|---------------------------|-------------|-------------|--------|
| swarm-manager | - | virtualbox | Running | tcp://192.168.99.100:2376 | v17.04.0-ce | v17.04.0-ce | |

Teraz należy wskazać nowego menedżera maszynie Docker Machine. Z komunikatów wyświetlonych w wyniku utworzenia menedżera Swarm wynika, że w celu wskazania węzła należy skorzystać z polecenia:

```
$ docker-machine env swarm-manager
```

Na ekranie pojawi się lista poleceń, które należy uruchomić w celu skonfigurowania lokalnego klienta Dockera, tak aby mógł komunikować się z utworzonym przed chwilą hostem Dockera. Oto konfiguracja mojego środowiska:

```
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/russ/.docker/machine/machines/swarmmanager"
export DOCKER_MACHINE_NAME="swarm-manager"
# W celu skonfigurowania powłoki skorzystaj z następującego polecenia:
# eval $(docker-machine env swarm-manager)
```

Na końcu tego komunikatu znajduje się polecenie, które należy uruchomić w celu wskazania menedżera Swarm:

```
$ eval $(docker-machine env swarm-manager)
```

Teraz na liście maszyn dostępnych z poziomu hosta powinniśmy znaleźć aktywną (patrz kolumna ACTIVE) maszynę `swarm-master`. W związku z tym możemy teraz uruchomić polecenie:

```
$ docker-machine ls
```

Spowoduje ono wyświetlenie następującej listy:

| 1. russ (bash) | | | | | | | |
|----------------|--------|------------|---------|---------------------------|-------|-------------|--------|
| NAME | ACTIVE | DRIVER | STATE | URL | SWARM | DOCKER | ERRORS |
| swarm-manager | * | virtualbox | Running | tcp://192.168.99.100:2376 | | v17.04.0-ce | |
| russ | in | | | | | | |

Uruchomiliśmy nasz pierwszy host. Teraz możemy dodać dwa węzły wykonawcze. W tym celu powinniśmy skorzystać z następującego polecenia uruchamiającego dwa kolejne hosty Dockera:

```
$ docker-machine create \
-d virtualbox \
swarm-worker01
$ docker-machine create \
-d virtualbox \
swarm-worker02
```

Po uruchomieniu dwóch kolejnych hostów ponownie sprawdzmy listę hostów:

```
$ docker-machine ls
```

Teraz lista ta powinna wyglądać tak:

| 1. russ (bash) | | | | | | | |
|----------------|--------|------------|---------|---------------------------|-------|-------------|--------|
| NAME | ACTIVE | DRIVER | STATE | URL | SWARM | DOCKER | ERRORS |
| swarm-manager | * | virtualbox | Running | tcp://192.168.99.100:2376 | | v17.04.0-ce | |
| swarm-worker01 | - | virtualbox | Running | tcp://192.168.99.101:2376 | | v17.04.0-ce | |
| swarm-worker02 | - | virtualbox | Running | tcp://192.168.99.102:2376 | | v17.04.0-ce | |
| russ | in | | | | | | |

Zwróćmy uwagę, że nie wykonaliśmy jeszcze żadnej czynności mającej na celu utworzenie klastra Swarm. Jak na razie uruchomiliśmy tylko hosty, na których uruchomiony zostanie klaster.

Lista wyświetlna po uruchomieniu polecenia docker-machine ls zawiera kolumnę SWARM. Dane w tej kolumnie są umieszczane tylko po uruchomieniu hostów Dockera za pomocą samodzielnego polecenia Docker Swarm, które jest wbudowane w maszynę Docker Machine.

Czas uruchomić menedżera Swarm. W tym celu należy przekazać do hosta kilka poleceń Docker Machine. Oto polecenie tworzące i uruchamiające menedżera:

```
$ docker $(docker-machine config swarm-manager) swarm init \
--advertise-addr $(docker-machine ip swarm-manager):2377 \
--listen-addr $(docker-machine ip swarm-manager):2377
```

Po uruchomieniu tego polecenia powinieneś zobaczyć komunikat podobny do poniższego:

```
Swarm initialized: current node (qha7m9bf55wwd8p3e0jiyk7yf) is now a manager.
To add a worker to this swarm, run the following command:
```

```
docker swarm join \
--token
SWMTKN-1-3fxwovyzh24120myqbzolpen303uzk562y26q4z1wgxto5avm3-4aiagep3e2a1nwy
af4yjp7ic5 \
192.168.99.100:2377
```

```
To add a manager to this swarm, run 'docker swarm join-token manager'
and follow the instructions.
```

Ze zwrotnego komunikatu wynika, że po zainicjowaniu menedżera dostaliśmy unikalny kod (token). Kod ten będzie musiał być przekazany przez węzły w celu przeprowadzenia procesu uwierzytelniania i podłączenia się do klastra.

Dołączanie wykonawców

W celu dodania dwóch wykonawców Swarm do klastra musimy uruchomić poniższe polecenia (uruchamiając je samodzielnie, pamiętaj o konieczności umieszczenia właściwego kodu Twojego menedżera):

```
$ docker $(docker-machine config swarm-worker01) swarm join \
$(docker-machine ip swarm-manager):2377 \
--token
SWMTKN-1-3fxwovyzh24120myqbzolpen303uzk562y26q4z1wgxto5avm3-4aiagep3e2a1nwyaf4yjp7ic5
```

Aby dodać drugiego wykonawcę, uruchom polecenie:

```
docker $(docker-machine config swarm-worker02) swarm join \
$(docker-machine ip swarm-manager):2377 \
--token
SWMTKN-1-3fxwovyzh24120myqbzolpen303uzk562y26q4z1wgxto5avm3-4aiagep3e2a1nwyaf4yjp7ic5
```

Po wykonaniu każdego z tych poleceń na ekranie powinien pojawić się komunikat potwierdzający dołączenie węzła do klastra:

```
This node joined a swarm as a worker.
```

Listy węzłów

W celu sprawdzenia obecności narzędzia Docker Swarm uruchom polecenie:

```
$ docker-machine ls
```

Sprawdź, czy Twój lokalny klient Dockera jest w dalszym ciągu skonfigurowany tak, aby łączył się z węzłem `swarm-manager`. Jeżeli nie jest, to uruchom ponownie polecenie:

```
$ eval $(docker-machine env swarm-manager)
```

Klient Dockera łączy się teraz z węzłem `swarm-manager`, możesz więc uruchomić polecenie:

```
$ docker node ls
```

Spowoduje ono nawiązanie połączenia z `swarm-master` i zwróci listę wszystkich węzłów klastra. Oto lista zawierająca wszystkie trzy węzły:

```
russ in ~
⚡ docker-machine ls
NAME      ACTIVE   DRIVER    STATE     URL
swarm-manager * virtualbox Running  tcp://192.168.99.100:2376
swarm-worker01 - virtualbox Running  tcp://192.168.99.101:2376
swarm-worker02 - virtualbox Running  tcp://192.168.99.102:2376
russ in ~
⚡ docker node ls
ID          HOSTNAME    STATUS  AVAILABILITY  MANAGER STATUS
qg3ycvryuccvsslo2cc4aa8r4  swarm-worker02  Ready  Active
qha7m9bf55wd8p3e0j1yk7yf *  swarm-manager   Ready  Active      Leader
wgtdfdnhcau7fc7xsj08uo7do  swarm-worker01  Ready  Active
russ in ~
⚡
```

Zarządzanie klastrem

Możemy już przejść do zagadnień związanych z zarządzaniem klastrami.

Hosty Swarm i uruchomione w nich kontenery mogą być zarządzane na dwa sposoby. Zaczniemy od opisu podstawowych związanych z nimi zagadnień.

Jak już wiesz, istnieje możliwość wyświetlenia listy węzłów wchodzących w skład klastra przy użyciu lokalnego klienta Dockera skonfigurowanego tak, aby łączył się z hostem zarządzającym. Przy takiej konfiguracji klienta Dockera możemy skorzystać z polecenia:

```
$ docker info
```

Wyświetli ono wiele informacji na temat hosta:

```
Containers: 0
    Running: 0
    Paused: 0
    Stopped: 0
Images: 0
Server Version: 17.04.0-ce
Storage Driver: aufs
Root Dir: /mnt/sda1/var/lib/docker/aufs
Backing Filesystem: extfs
Dirs: 0
Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
    Volume: local
    Network: bridge host macvlan null overlay
Swarm: active
    NodeID: qha7m9bf55wwd8p3e0jiyk7yf
    Is Manager: true
    ClusterID: n5akyh6xsnc15qnx5ccp54vrr
    Managers: 1
    Nodes: 3
    Orchestration:
        Task History Retention Limit: 5
    Raft:
        Snapshot Interval: 10000
        Number of Old Snapshots to Retain: 0
        Heartbeat Tick: 1
        Election Tick: 3
    Dispatcher:
        Heartbeat Period: 5 seconds
    CA Configuration:
        Expiry Duration: 3 months
    Node Address: 192.168.99.100
    Manager Addresses:
        192.168.99.100:2377
Runtimes: runc
Default Runtime: runc
Init Binary:
containerd version: 422e31ce907fd9c3833a38d7b8fdd023e5a76e73
runc version: 9c2d8d184e5da67c95d601382adf14862e4f2228
init version: 949e6fa
Security Options:
    seccomp
        Profile: default
Kernel Version: 4.4.59-boot2docker
Operating System: Boot2Docker 17.04.0-ce (TCL 7.2); HEAD : c69677f - Thu
Apr 6 16:26:16 UTC 2017
OSType: linux
Architecture: x86_64
```

```

CPUs: 1
Total Memory: 995.8 MiB
Name: swarm-manager
ID: VKLO:MKJK:Y4UD:2IXV:WBA3:LTZE:J4MU:MGAD:VF7Z:QVVI:XNQG:SMAB
Docker Root Dir: /mnt/sda1/var/lib/docker
Debug Mode (client): false
Debug Mode (server): true
  File Descriptors: 32
  Goroutines: 149
  System Time: 2017-04-26T09:51:29.683890515Z
  EventsListeners: 0
Username: russmckendrick
Registry: https://index.docker.io/v1/
Labels:
  provider=virtualbox
Experimental: false
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false

```

Sekcja Swarm zawiera informacje o klastrze. Niestety polecenie `docker info` może być użyte tylko w kontekście hosta, z którym nasz klient może się obecnie komunikować. Na szczęście polecenie `docker node` jest świadomie istnienia klastra, a więc możemy z niego skorzystać w celu uzyskania informacji o każdym węźle klastra:

```
$ docker node inspect swarm-manager --pretty
```

Dodanie flagi `--pretty` do polecenia `docker node inspect` sprawi, że zwrócone przez nie informacje będą w czytelnej formie. W przypadku pominięcia flagi `--pretty` Docker zwróci surowy obiekt JSON zawierający wyniki zapytania wygenerowanego przez polecenie `inspect`, które zostało wysłane do klastra.

Powinniśmy uzyskać następujące informacje opisujące naszego menedżera Swarm:

```

ID: qha7m9bf55wwd8p3e0jiyk7yf
Labels:
Hostname: swarm-manager
Joined at: 2017-04-25 16:56:47.092119605 +0000 utc
Status:
  State: Ready
  Availability: Active
  Address: 192.168.99.100
Manager Status:
  Address: 192.168.99.100:2377
  Raft Status: Reachable
  Leader: Yes
Platform:
  Operating System: linux
  Architecture: x86_64
Resources:
  CPUs: 1

```

```
Memory: 995.8 MiB
Plugins:
  Network: bridge, host, macvlan, null, overlay
  Volume: local
Engine Version: 17.04.0-ce
Engine Labels:
  - provider = virtualbox
```

Spróbujmy uruchomić to samo polecenie jeszcze raz, ale tym razem skierujmy je do jednego z węzłów roboczych:

```
$ docker node inspect swarm-worker01 --pretty
```

Zwrócone zostaną podobne informacje:

```
ID: wgtfdnhcau7fcr7xsj08uo7do
Labels:
Hostname: swarm-worker01
Joined at: 2017-04-25 17:11:03.532507218 +0000 utc
Status:
  State: Ready
  Availability: Active
  Address: 192.168.99.101
Platform:
  Operating System: linux
  Architecture: x86_64
Resources:
  CPUs: 1
  Memory: 995.8 MiB
Plugins:
  Network: bridge, host, macvlan, null, overlay
  Volume: local
Engine Version: 17.04.0-ce
Engine Labels:
  - provider = virtualbox
```

W zwróconych danych brakuje informacji na temat stanu funkcjonalności menedżera. Wynika to z tego, że węzły robocze nie muszą dysponować informacjami dotyczącymi statusu węzłów zarządzających — do pracy wystarcza im wiedza o tym, że mogą otrzymywać instrukcje od menedżerów.

W zwróconych danych opisujących hosta znajdują się informacje na temat liczby kontenerów, liczby obrazów, dostępnej mocy obliczeniowej i pamięci, a także innych interesujących aspektów hosta.

Promowanie hosta roboczego

Załóżmy, że chcesz przeprowadzić pewne prace związane z utrzymaniem pojedynczego węzła menedżera, ale jednocześnie musisz zachować dostępność klastra. Nie stanowi to problemu, ponieważ możesz dokonać promocji węzła roboczego na węzeł menedżera.

Dysponujemy aktywnym lokalnym klastrem składającym się z trzech węzłów, a więc możemy promować hosta `swarm-worker01` na nowego menedżera. W tym celu należy uruchomić polecenie:

```
$ docker node promote swarm-worker01
```

Od razu po uruchomieniu tego polecenia powinien pojawić się na ekranie komunikat potwierdzający promocję węzła:

```
Node swarm-worker01 promoted to a manager in the swarm.
```

Wyświetlmy listę węzłów za pomocą polecenia:

```
$ docker node ls
```

Teraz w kolumnie `MANAGER STATUS` (status menedżera) powinny się znaleźć informacje opisujące status dwóch węzłów:

| ID | HOSTNAME | STATUS | AVAILABILITY | MANAGER STATUS |
|----------------------------|----------------|--------|--------------|----------------|
| qg3ycvryuccvsslo2cc4aa8r4 | swarm-worker02 | Ready | Active | |
| qha7m9bf55wd8p3e0j1yk7yf * | swarm-manager | Ready | Active | Leader |
| wgtfdnhcau7fc7xsj08u07do | swarm-worker01 | Ready | Active | Reachable |

Węzeł `swarm-manager` jest w dalszym ciągu głównym węzłem zarządzającym. Spróbujmy to zmienić.

Degradacja węzła menedżera

Dysponujesz już dwoma węzłami, które mogą pełnić funkcję menedżera. W celu zdegradowania obecnego menedżera do roli węzła roboczego wystarczy uruchomić polecenie:

```
$ docker node demote swarm-manager
```

Po uruchomieniu polecenia od razu zostaniemy poinformowani o wykonaniu operacji:

```
Manager swarm-manager demoted in the swarm.
```

Właśnie dokonaliśmy degradacji jednego z węzłów. Sprawdźmy status węzłów klastra za pomocą polecenia:

```
$ docker node ls
```

Lokalny klient Dockera wciąż kieruje zapytania do zdegradowanego przed chwilą węzła, a więc zwrócony zostanie następujący komunikat:

```
Error response from daemon: This node is not a swarm manager. Worker nodes can't be used to view or modify cluster state. Please run this command on a manager node or promote the current node to a manager.
```

Przypominamy, że aktualizację konfiguracji lokalnego klienta tak, aby komunikował się z innymi węzłami, można przeprowadzić łatwo za pomocą narzędzia Docker Machine. W celu wskazania nowego węzła menedżera należy uruchomić polecenie:

```
$ eval $(docker-machine env swarm-worker01)
```

Klient Dockera komunikuje się z nowym menedżerem, a więc możemy uruchomić polecenie:

```
$ docker node ls
```

Zwróci ono następującą listę węzłów:

| ID | HOSTNAME | STATUS | AVAILABILITY | MANAGER STATUS |
|---------------------------|----------------|--------|--------------|----------------|
| qg3ycvryuccvsslo2cc4aa8r4 | swarm-worker02 | Ready | Active | |
| qha7m9bf55wd8p3e01yk7yf | swarm-manager | Ready | Active | |
| wgtfdnhcau7fcr7xsj08uo7do | * | Ready | Active | Leader |

Drenaż węzła

Aby usunąć tymczasowo węzeł z klastra (w celu wykonania prac związanych z utrzymaniem oprogramowania) musimy zmienić status węzła na *Drain* (drenowanie). Zróbmy to na przykładzie naszego poprzedniego węzła menedżera. W tym celu musimy uruchomić polecenie:

```
$ docker node update --availability drain swarm-manager
```

Spowoduje ono zatrzymanie wszystkich nowych zadań, takich jak uruchamianie nowych kontenerów, a także zadań wykonywanych obecnie przez drenowany węzeł. Po zablokowaniu nowych zadań wszystkie aktywne zadania zostaną przeniesione z drenowanego węzła do węzłów o statusie *ACTIVE* (aktywny).

Z poniższej listy węzłów wynika, że drenowany jest węzeł *swarm-manager* — nadano mu status *Drain* (patrz kolumna *AVAILABILITY* — dostępność):

| ID | HOSTNAME | STATUS | AVAILABILITY | MANAGER STATUS |
|---------------------------|----------------|--------|--------------|----------------|
| qg3ycvryuccvsslo2cc4aa8r4 | swarm-worker02 | Ready | Active | |
| qha7m9bf55wd8p3e01yk7yf | swarm-manager | Ready | Drain | |
| wgtfdnhcau7fcr7xsj08uo7do | * | Ready | Active | Leader |

Gdy węzeł nie przyjmuje nowych zadań, a wszystkie aktywne zadania zostały przeniesione na dwa pozostałe węzły, możemy wykonać w sposób bezpieczny wszystkie prace związane z utrzymaniem oprogramowania, takie jak np. ponowne uruchomienie hosta. W celu ponownego uruchomienia hosta *swarm-manager* musimy uruchomić dwa poniższe polecenia po

upewnieniu się, że jesteśmy połączeni z hostem Dockera (powinniśmy widzieć baner *boot2docker*, taki jak ten, który pokazano na zrzucie znajdującym się pod poleceniami):

```
$ docker-machine ssh swarm-manager
$ sudo reboot
```

Po ponownym uruchomieniu hosta skorzystaj z polecenia:

```
$ docker node ls
```

W kolumnie **AVAILABILITY** powinien teraz znajdować się zapis *Pause* (pauza). W celu doda-
nia węzła z powrotem do klastra uruchom poniższe polecenie. Spowoduje ono zmianę stanu
zapisanego w kolumnie **AVAILABILITY** z *Pause* na *Active* (aktywny):

```
$ docker node update --availability active swarm-manager
```

Z wyświetlonych informacji wynika, że węzeł jest już aktywny, a więc może wykonywać nowe zadania:

| ID | HOSTNAME | STATUS | AVAILABILITY | MANAGER STATUS |
|---------------------------|----------------|--------|--------------|----------------|
| qg3ycvryuccvsslo2cc4aa8r4 | swarm-worker02 | Ready | Active | |
| qha7m9bf55wwd8p3e0j1yk7yf | swarm-manager | Ready | Pause | |
| wgtfdnhcau7fcr7xsj08uo7do | * | Ready | Active | Leader |

Opisaliśmy zagadnienia związane z tworzeniem klastrów Docker Swarm i zarządzaniem nimi,
a więc możemy przejść do uruchamiania zadań takich jak tworzenie i skalowanie usług.

Usługi i stosy Docker Swarm

Dotychczas korzystaliśmy z następujących poleceń:

```
$ docker swarm <polecenie>
$ docker node <polecenie>
```

Te dwie grupy poleceń umożliwiają uruchamianie klastra Docker Swarm składającego się z utworzonych wcześniej hostów Dockera i zarządzanie nim. W tym podrozdziale przyjrzymy się dwóm kolejnym grupom poleceń:

```
$ docker service <polecenie>
$ docker stack <polecenie>
```

Polecenia service i stack pozwalają na wykonywanie zadań polegających na uruchamianiu kontenerów wchodzących w skład klastra oraz skalowaniu ich i zarządzaniu nimi.

Usługi

Polecenie service umożliwia uruchamianie kontenerów korzystających z klastra Swarm. Przyjrzymy się procesowi uruchamiania bardzo prostej, jednokontenerowej usługi w klastrze Swarm. W celu uruchomienia jej w naszym klastrze skorzystajmy z polecenia:

```
$ docker service create --name cluster --constraint "node.role == worker" -
→p:80:tcp russmckendrick/cluster
```

Polecenie to utworzy usługę o nazwie cluster składającą się z jednego kontenera, którego port 80 będzie mapowany do hosta. Usługa ta będzie działała tylko w węzłach, które pełnią funkcję węzłów roboczych.

Zanim przyjrzymy się zastosowaniom usługi, sprawdźmy, czy działa ona w naszej przeglądarce. W tym celu musimy ustalić adres IP naszych dwóch kontenerów roboczych. Zaczniemy od sprawdzenia tego, które z węzłów są węzłami wykonawczymi. Zróbcmy to za pomocą polecenia:

```
$ docker-machine ls
```

Znamy funkcje poszczególnych węzłów. Teraz możemy ustalić ich adresy IP za pomocą polecenia:

```
$ docker-machine ls
```

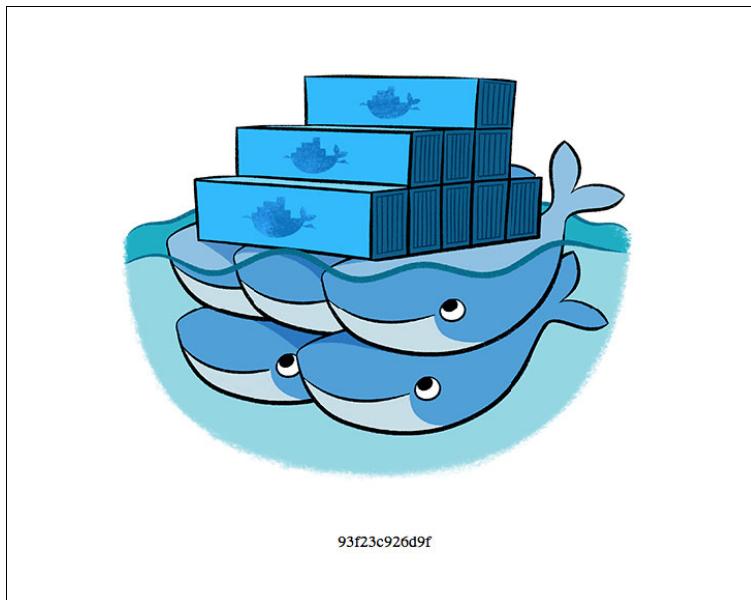
Przyjrzymy się zwróconym danym:

| ID | HOSTNAME | STATUS | AVAILABILITY | MANAGER STATUS |
|--------------------------|----------------|----------------|--------------|----------------|
| qg3ycvryuccsslo2cc4aa8r4 | swarm-worker02 | Ready | Active | |
| qha79bt55wd8p3e0j1yk7yf | swarm-manager | Ready | Active | |
| wgtfdnhcau7fc7xsj08uo7do | * | swarm-worker01 | Ready | Active |

| NAME | ACTIVE | DRIVER | STATE | URL | SWARM | DOCKER | ERRORS |
|----------------|--------|------------|---------|---------------------------|-------------|-------------|--------|
| swarm-manager | - | virtualbox | Running | tcp://192.168.99.100:2376 | v17.04.0-ce | v17.04.0-ce | |
| swarm-worker01 | * | virtualbox | Running | tcp://192.168.99.101:2376 | v17.04.0-ce | v17.04.0-ce | |
| swarm-worker02 | - | virtualbox | Running | tcp://192.168.99.102:2376 | v17.04.0-ce | v17.04.0-ce | |

Nasze węzły robocze to *swarm-manager* i *swarm-worker02*. Ich adresy IP to *192.168.99.100* i *192.168.99.102*.

Wpisanie któregokolwiek z tych adresów IP w przeglądarce (*http://192.168.99.100/* lub *http://192.168.99.102/*) spowoduje wyświetlenie strony wygenerowanej przez aplikację *russmckendrick/cluster*, która zawiera grafikę Docker Swarm i nazwę hosta kontenera, z którego wyświetlono stronę:



Po uruchomieniu usługi w klastrze możemy spróbować wyświetlić informacje o niej. Zaczniemy od polecenia:

```
$ docker service ls
```

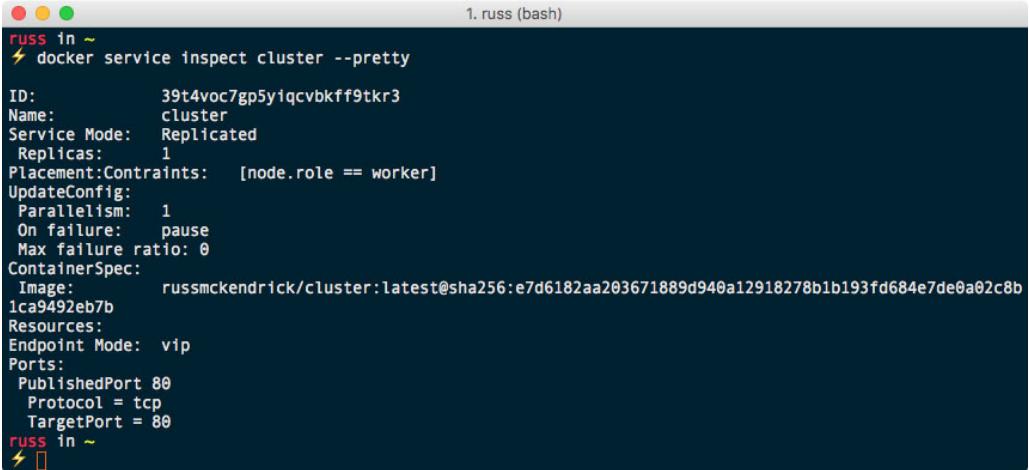
W naszym przypadku wyświetlona lista zawiera jedną usługę o nazwie *cluster*:

```
russ in ~
⚡ docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
39t4voc7gp5y  cluster  replicated  1/1      russmckendrick/cluster:latest
russ in ~
⚡
```

Jak widzisz, usługa ta jest odtwarzalna (*replicated*), a kontener *1/1* jest aktywny. Teraz możemy poszukać bardziej szczegółowych informacji o usłudze za pomocą polecenia *inspect*:

```
$ docker service inspect cluster --pretty
```

Na ekranie pojawią się szczegółowe informacje o usłudze:



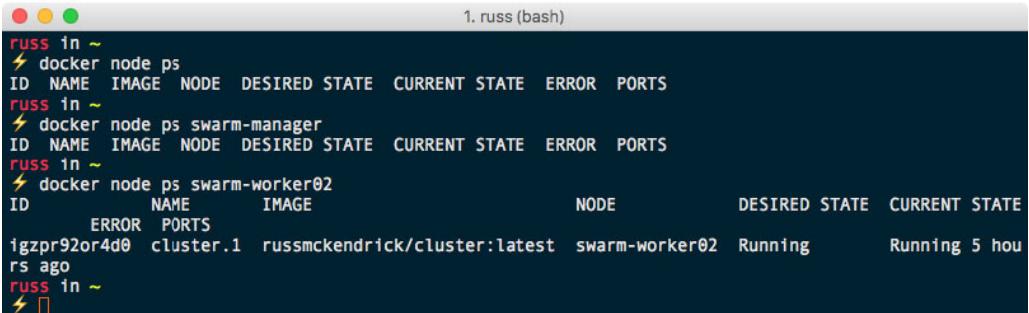
```
1. russ (bash)
russ in ~
⚡ docker service inspect cluster --pretty
ID:          39t4voc7gp5yiqcvbkff9tkr3
Name:        cluster
Service Mode: Replicated
Replicas:    1
Placement:Constraints:  [node.role == worker]
UpdateConfig:
  Parallelism: 1
  On failure:  pause
  Max failure ratio: 0
ContainerSpec:
  Image:      russmckendrick/cluster:latest@sha256:e7d6182aa203671889d940a12918278b1b193fd684e7de0a02c8b
  1ca9492eb7b
  Resources:
  Endpoint Mode: vip
  Ports:
    PublishedPort 80
      Protocol = tcp
      TargetPort = 80
russ in ~
⚡
```

Dotychczas nie przejmowaliśmy się tym, na którym z węzłów roboczych działa usługa. To dość ważna funkcja narzędzia Docker Swarm — dzięki niej nie musimy martwić się rozmieszczeniem poszczególnych kontenerów.

Zanim zajmiemy się skalowaniem usługi, sprawdźmy, na którym hoście działa nasz pojedynczy kontener. W tym celu należy uruchomić następujące polecenia:

```
$ docker node ps
$ docker node ps swarm-manager
$ docker node ps swarm-worker02
```

Polecenia te wyświetlą listy kontenerów uruchomionych w każdym z naszych hostów. Polecenie docker node ps domyślnie wyświetla kontenery uruchomione w docelowym hoście (w moim przypadku jest to host *swarm-worker01*):



```
1. russ (bash)
russ in ~
⚡ docker node ps
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS
russ in ~
⚡ docker node ps swarm-manager
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS
russ in ~
⚡ docker node ps swarm-worker02
ID           NAME          IMAGE          NODE          DESIRED STATE  CURRENT STATE
          ERROR PORTS
igzpr92or4d0  cluster.1  russmckendrick/cluster:latest  swarm-worker02  Running      Running  5 hou
rs ago
russ in ~
⚡
```

Przyjrzyjmy się skalowaniu naszej usługi do sześciu egzemplarzy jej kontenera. Poniższe polecenia skalują i sprawdzają stan usługi:

```
$ docker service scale cluster=6
$ docker service ls
$ docker node ps swarm-manager
$ docker node ps swarm-worker02
```

Sprawdzamy stan tylko dwóch węzłów, ponieważ początkowo zadeklarowaliśmy, że usługa ma być uruchamiana tylko w węzłach roboczych. Z informacji wyświetlonych w poniższym oknie wynika, że w każdym z dwóch węzłów roboczych działają po trzy kontenery:

```
russ in ~
⚡ docker service scale cluster=6
cluster scaled to 6
russ in ~
⚡ docker service ls
ID          NAME      MODE      REPLICAS  IMAGE
39t4voc7gp5y  cluster  replicated  6/6        russmckendrick/cluster:latest
russ in ~
⚡ docker node ps swarm-manager
ID          NAME      IMAGE
NODE      DESIRED STATE  CURRENT STATE
ERROR PORTS
07u4mqksfnb  cluster.3  russmckendrick/cluster:latest  swarm-manager  Running   Running 43 sec
on 3 days ago
rtczz50j0n30  cluster.4  russmckendrick/cluster:latest  swarm-manager  Running   Running 43 sec
on 3 days ago
9zdgdyn3i1ol  cluster.6  russmckendrick/cluster:latest  swarm-manager  Running   Running 43 sec
on 3 days ago
russ in ~
⚡ docker node ps swarm-worker02
ID          NAME      IMAGE
NODE      DESIRED STATE  CURRENT STATE
ERROR PORTS
igzpr92or4d0  cluster.1  russmckendrick/cluster:latest  swarm-worker02  Running   Running 5 hours ago
80ggm7itwrkk  cluster.2  russmckendrick/cluster:latest  swarm-worker02  Running   Running about a minute ago
d2dhcwzzhh3   cluster.5  russmckendrick/cluster:latest  swarm-worker02  Running   Running about a minute ago
russ in ~
⚡
```

Zanim przyjrzymy się stosom, usuńmy nasze usługi. W tym celu należy uruchomić polecenie:

```
$ docker service rm cluster
```

Usunie ono wszystkie kontenery, pozostawiając w hostach pobrane obrazy.

Stosy

Narzędzie Swarm wraz z usługami umożliwia tworzenie złożonych, wysoce dostępnych, wielokontenerowych aplikacji. W przypadku klastrów niekorzystających z technologii Swarm ręczne uruchamianie wielu kontenerów wchodzących w skład aplikacji może być zbyt pracochłonne i trudne. Aby rozwiązać ten problem, Docker umożliwia definiowanie usług w plikach Docker Compose.

Oto zawartość pliku Docker Compose tworzącego aplikację, którą uruchamialiśmy w poprzedniej sekcji:

```
version: "3"
services:
  cluster:
    image: russmckendrick/cluster
    ports:
      - "80:80"
    deploy:
      replicas: 6
      restart_policy:
        condition: on-failure
      placement:
        constraints:
          - node.role == worker
```

Jak widzisz, stos może składać się z wielu usług zdefiniowanych w sekcji `services` pliku Docker Compose.

Poza normalnymi poleceniami Docker Compose do tego pliku możesz dodać również sekcję wdrażania (`deploy`). Sekcja ta definiuje wszystko, co jest związane z elementem Swarm stosu. W poprzednim przykładzie tworzyliśmy sześć replik umieszczonych w dwóch węzłach roboczych. Ponadto aktualizowaliśmy domyślne zasady ponownego uruchamiania, co mogłeś z obserwować podczas analizy usługi z poprzedniej sekcji (była ona wstrzymana — *paused*). Według zmodyfikowanych zasad kontener jest uruchamiany ponownie zawsze wtedy, gdy przestanie odpowiadać.

W celu aktywowania naszego stosu musimy skopiować zaprezentowany wcześniej kod do pliku o nazwie `docker-compose.yml`, a następnie uruchomić polecenie:

```
$ docker stack deploy --compose-file=docker-compose.yml cluster
```

Docker, podobnie jak w przypadku uruchamiania kontenerów za pomocą narzędzia Docker Compose, utworzy nową sieć, a następnie uruchomi w niej usługi.

Status stosu można sprawdzić za pomocą polecenia:

```
$ docker stack ls
```

Ze zwróconych danych będzie wynikało, że utworzono pojedynczą usługę. W celu wyświetlenia szczegółowych informacji na temat usługi utworzonej przez stos należy skorzystać z polecenia:

```
$ docker stack services cluster
```

Aby sprawdzić miejsca, w których działają kontenery wchodzące w skład stosu, uruchom polecenie:

```
$ docker stack ps cluster
```

Przyjrzyj się danym wyświetlonym w oknie Terminala:

```
russ in ~/Desktop/cluster
⚡ docker stack ls
NAME      SERVICES
cluster   1

russ in ~/Desktop/cluster
⚡ docker stack services cluster
ID          NAME      MODE      REPLICAS  IMAGE
ty0czqwd47lc  cluster_cluster  replicated  6/6        russmckendrick/cluster:latest
russ in ~/Desktop/cluster
⚡ docker stack ps cluster
ID          NAME      IMAGE      NODE      DESIRED STATE  CURRENT STATE
NT STATE      ERROR PORTS
07wrabg5dso2  cluster_cluster.1  russmckendrick/cluster:latest  swarm-manager  Running   Running
ng 8 minutes ago
onsslc92pos1  cluster_cluster.2  russmckendrick/cluster:latest  swarm-worker02  Running   Running
ng 8 minutes ago
fbb2sk6butsy  cluster_cluster.3  russmckendrick/cluster:latest  swarm-manager  Running   Running
ng 8 minutes ago
v2l2ngznjfo1  cluster_cluster.4  russmckendrick/cluster:latest  swarm-worker02  Running   Running
ng 8 minutes ago
m0qvc5mth666  cluster_cluster.5  russmckendrick/cluster:latest  swarm-manager  Running   Running
ng 8 minutes ago
ujkex1f8uv2d  cluster_cluster.6  russmckendrick/cluster:latest  swarm-worker02  Running   Running
ng 8 minutes ago
russ in ~/Desktop/cluster
⚡
```

Dostęp do stosu możemy uzyskać za pomocą adresu IP węzłów (zostaniemy przekierowani do jednego z działających kontenerów).

W celu usunięcia stosu wystarczy uruchomić polecenie:

```
$ docker stack rm cluster
```

Usunie ono wszystkie usługi i sieci utworzone podczas uruchamiania stosu.

Kasowanie klastra Swarm

W kolejnej sekcji nie będziemy potrzebować klastra, a więc możemy go skasować za pomocą polecenia:

```
$ docker-machine rm swarm-manager swarm-worker01 swarm-worker02
```

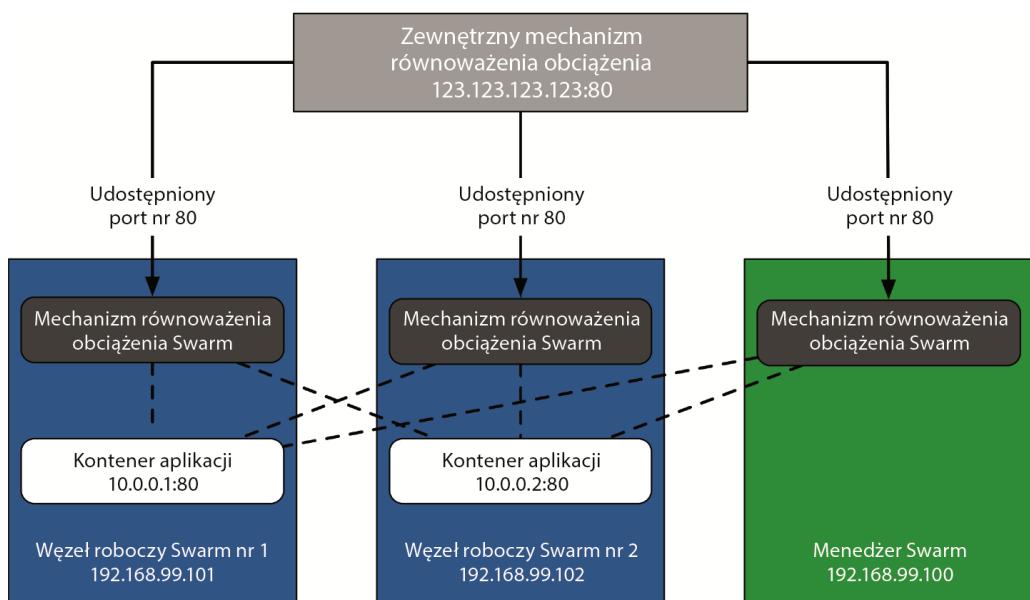
W przypadku potrzeby ponownego uruchomienia klastra Swarm wystarczy, że wykonasz od początku polecenia zaprezentowane w tym rozdziale.

Równoważenie obciążeń, nakładki i tworzenie harmonogramów

W poprzednich sekcjach zajmowaliśmy się uruchamianiem usług i stosów. W celu uzyskania dostępu do uruchomionych aplikacji mogliśmy korzystać z dowolnego adresu IP hosta wchodzącego w skład klastra. Jak to możliwe?

Równoważenie obciążień wejściowych

Docker Swarm ma wbudowany moduł równoważenia obciążień wejściowych, który ułatwia rozdzielenie ruchu pomiędzy dostępnymi publicznie kontenerami. W związku z tym istnieje możliwość udostępniania aplikacjom wchodzącym w skład klastra Swarm dostępu do usług takich jak zewnętrzne mechanizmy równoważenia obciążenia (np. Amazon Elastic Load Balancer) i zachowania pewności co do tego, że żądania zostaną przekazane do właściwych kontenerów, niezależnie od tego, co dzieje się z hostem, na którym się one obecnie znajdują. Mechanizm ten przedstawiono na poniższym diagramie:



Dzięki temu rozwiązaniu możliwe jest skalowanie, aktualizowanie i naprawianie aplikacji bez potrzeby zmiany konfiguracji zewnętrznego mechanizmu równoważącego obciążenie.

Nakłady sieciowe

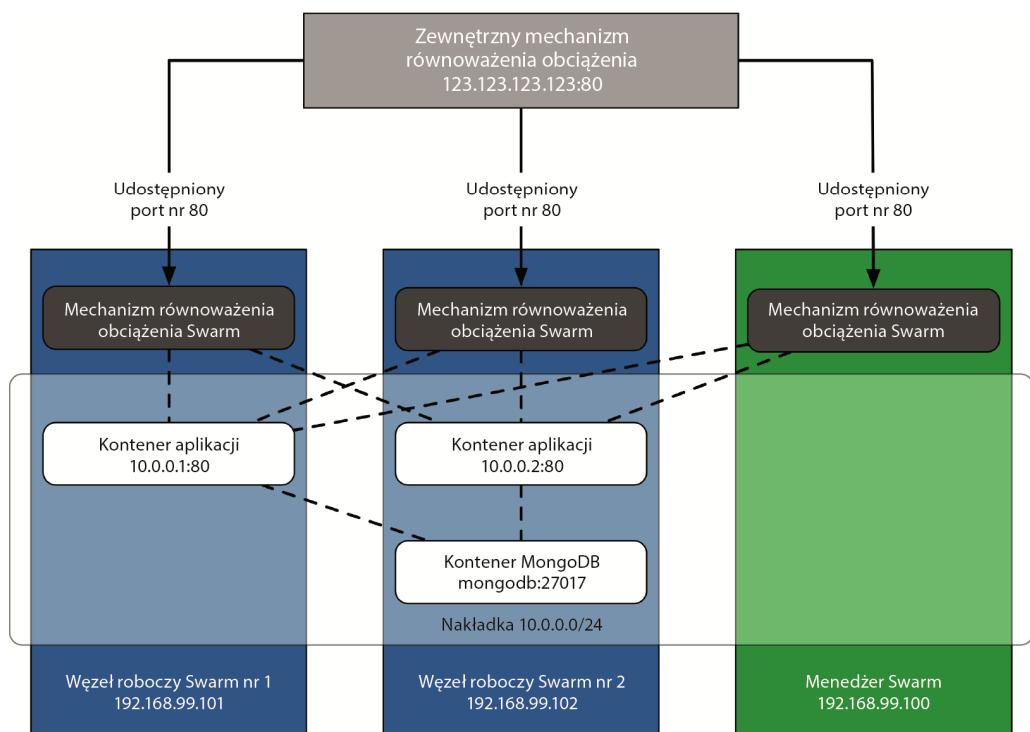
W przytoczonym przykładzie uruchomiliśmy prostą usługę obsługującą pojedynczą aplikację. A co, gdybyśmy chcieli dodać do tej aplikacji warstwę bazy danych, która znajduje się zwykle w określonym miejscu sieci? Jak można to zrobić?

Warstwa nakładek sieciowej Docker Swarm rozszerza sieć, w której uruchamiane są kontenery znajdujące się w różnych hostach, a więc możliwe jest uruchomienie poszczególnych usług i stosów we własnych, odizolowanych sieciach. W związku z tym kontener bazy danych MongoDB może być dostępny dla wszystkich kontenerów (niezależnie od hosta, na którym są uruchomione) korzystających z tej samej nakładki sieci na porcie nr 27017.

Być może myślisz sobie: „Zaraz, zaraz, czy w związku z tym muszę wpisywać do konfiguracji mojej aplikacji sztywny adres IP?”. Takie rozwiązanie nie sprawdziłoby się najlepiej w kontekście problemów, które ma rozwiązywać narzędzie Docker Swarm, a więc nie musisz sztywno określać wspomnianego adresu IP.

Każda nakłada sieci ma własną usługę DNS, a więc każdy kontener uruchomiony w sieci może ustalić adresy IP innych kontenerów znajdujących się w sieci na podstawie ich nazw. W związku z tym, konfigurując aplikację tak, aby łączyła się z instancją bazy danych, możemy skorzystać z adresu kontenera bazy MongoDB w formie `mongodb:27017`.

Nasz diagram będzie więc wyglądał następująco:



Korzystając z tego rozwiązania, należy wziąć pod uwagę jeszcze kilka czynników, które opiszemy w rozdziale 12. „Przepływy zadań w platformie Docker”.

Tworzenie harmonogramu

W momencie pisania tej książki Docker Swarm obsługuje tylko jedną strategię tworzenia harmonogramów. Rozwiązanie to nazywa się **Spread**. Pozwala ono na planowanie zadań wykonywanych na najmniej obciążonym węźle spełniającym wymagania określone w momencie uruchamiania usługi lub stosu. W większości przypadków nie ma potrzeby obarczania usług zbyt dużą liczbą ograniczeń.

Docker Swarm nie obsługuje obecnie jedynie powinowactwa, ale ograniczenie to można obejść, korzystając z ograniczeń. Staraj się nie komplikować zbytnio ograniczeń, ponieważ zdefiniowanie zbyt wielu ograniczeń podczas tworzenia usług może doprowadzić do problematycznego przeciążania niektórych hostów.

Podsumowanie

W tym rozdziale opisaliśmy zagadnienia związane z Docker Swarm. Zajęliśmy się instalacją tego narzędzia i jego komponentów. Przedstawiliśmy operacje łączenia węzłów, wyświetlania list węzłów, a także zarządzania węzłami zarządzającymi i roboczymi. Opisaliśmy polecenia związane z obsługą usługi i stosu, a także przedstawiliśmy wbudowany mechanizm równoważenia obciążień wejściowych, tworzenia nakładek sieci i harmonogramów.

W kolejnych trzech rozdziałach przedstawimy aplikacje przeznaczone do zarządzania hostami Dockera, kontenerami i obrazami wyposażone w graficzny interfejs użytkownika. Narzędzia te dysponują dużymi możliwościami i zdecydowanie się na wybór jednego z nich może być trudne. W związku z tym opiszemy je dokładnie. Zaczniemy od narzędzia Portainer.

Portainer

W tym rozdziale opiszemy zagadnienia związane z narzędziem Portainer, które umożliwia zarządzanie zasobami Dockera za pomocą interfejsu sieciowego. Poruszymy następujące tematy:

- historia prac nad narzędziem Portainer,
- uruchamianie narzędzia Portainer,
- korzystanie z narzędzia Portainer,
- Portainer i Docker Swarm.

Historia prac nad narzędziem Portainer

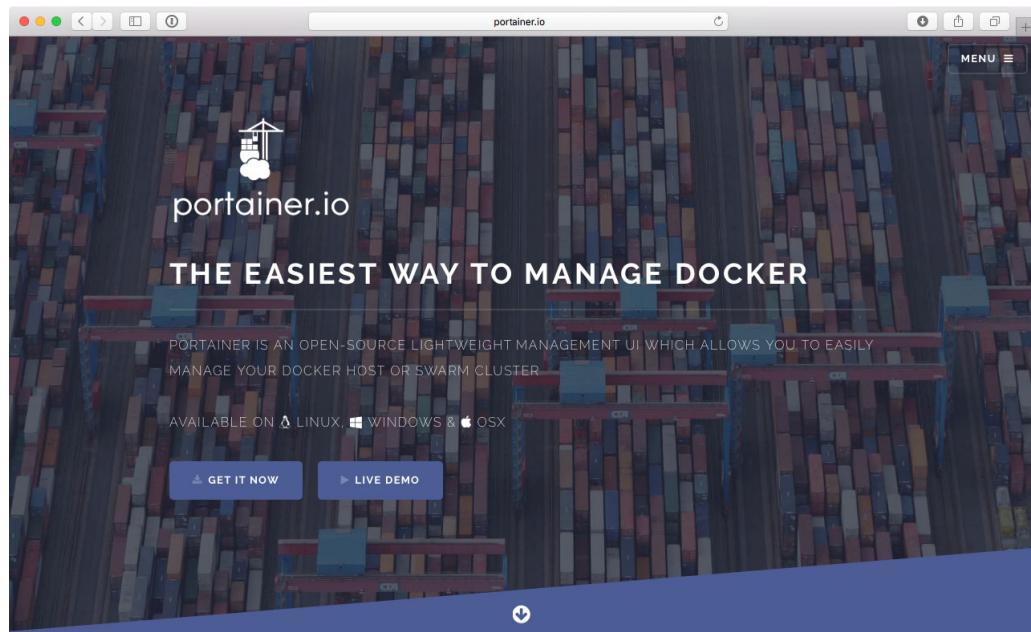
Zanim zakaszemy rękawy i zabierzmy się za instalowanie i używanie narzędzia Portainer, powinniśmy się przyjrzeć historii tego projektu. W pierwszym wydaniu tej książki pisaliśmy o projekcie Docker IU, którego autorem był Michael Crosby. Po okolo roku pracy przekazał on ten projekt Kevanowi Ahlquistowi. Z powodu sporu o nazwę zmieniono ją na UI for Docker.

Prace nad tym projektem trwały do momentu przyśpieszenia wprowadzania do jądra Docker Engine funkcji takich jak tryb Swarm. Mniej więcej w tym czasie projekt UI for Docker został wciągnięty do projektu, któremu wkrótce nadano nazwę Portainer. Jego pierwszą wersję upubliczniono w czerwcu 2016 r.

Twórcy narzędzia Portainer twierdzą, że od czasu upublicznienia jego pierwszej wersji zmodyfikowali około 70% jego kodu. W połowie 2017 r. dodano do niego nowe funkcje, takie jak sterowanie oparte na rolach i obsługa narzędzia Docker Compose. Twórcy narzędzia Portainer zakładają, że do końca 2017 r. uda im się przepisać kod odziedziczony po projekcie UI for Docker.

W grudniu 2016 r. w repozytorium GitHub UI for Docker opublikowano komunikat, że projekt ten nie będzie rozwijany i zamiast z niego należy korzystać z narzędzia Portainer.

Oto strona narzędzia Portainer znajdująca się pod adresem <https://www.portainer.io>:



Nie będziemy powtarzać instrukcji znajdujących się w sekcji *GET IT NOW* (zacznię pracę już teraz). Przedstawimy ich nieco zmodyfikowaną i poprawioną wersję.

Uruchamianie narzędzia Portainer

Zacznijmy od używania narzędzia Portainer do zarządzania pojedynczą instancją Dockera uruchomioną w środowisku lokalnym. Korzystam z systemu macOS, a więc opisywany przeze mnie proces będzie przebiegał w tym środowisku, ale moje instrukcje powinny również działać w innych środowiskach.

Zacznijmy od pobrania obrazu kontenera z repozytorium Docker Hub za pomocą następujących poleceń:

```
$ docker image pull portainer/portainer  
$ docker image ls
```

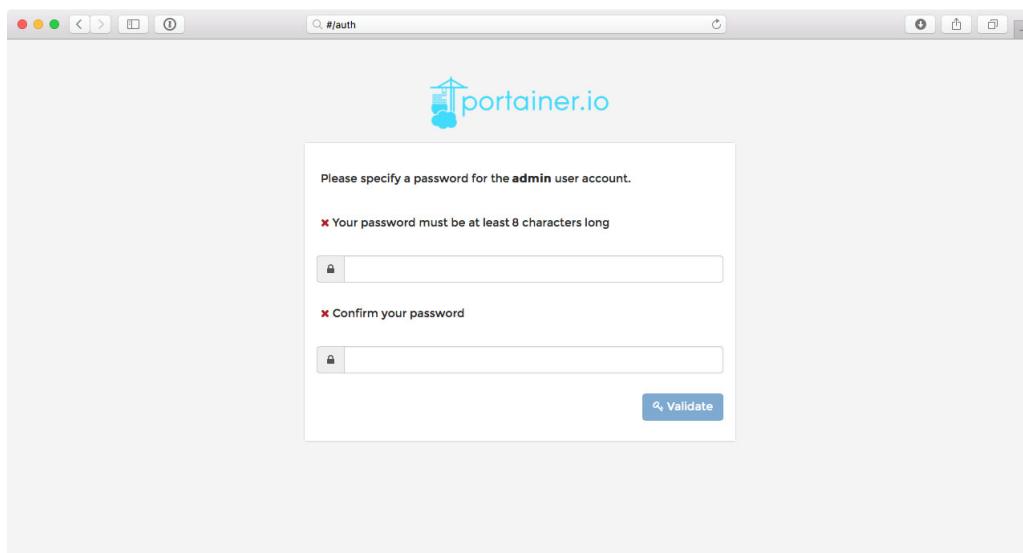
Według informacji zwróconych przez polecenie `docker image ls` obraz narzędzia Portainer zajmuje zaledwie 9,96 MB. W celu uruchomienia tego narzędzia skorzystajmy z polecenia:

```
$ docker container run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock
→portainer/portainer
```

Z uruchomionego przed chwilą polecenia wynika, że przeprowadzamy operacje montowania pliku *Docker Engine socket* w hoście Docker Host. Zabieg ten umożliwia uzyskanie przez narzędzie Portainer pełnego, nieograniczonego dostępu do silnika Docker Engine hosta. Dostęp ten jest niezbędny do zarządzania hostem. Pamiętaj o tym, że kontener Portainer ma pełny dostęp do hosta, a więc zachowaj ostrożność, dając komuś do niego dostęp, a także udostępniając go w zdalnych hostach.

W przypadku najprostszej instalacji nie musimy wykonywać żadnych dodatkowych operacji w wierszu poleceń, ale w celu zakończenia instalacji powinniśmy otworzyć przeglądarkę i wykonać w niej kolejne kroki. W związku z tym wejdź pod adres <http://localhost:9000/>.

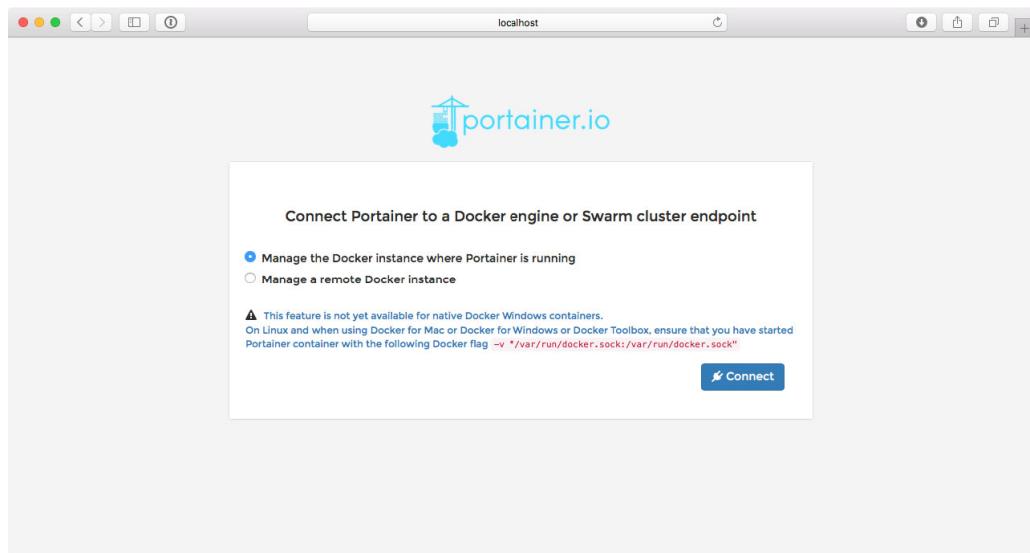
Pierwszy ekran powitalny poprosi Cię o określenie hasła administratora:



Po określeniu hasła zostaniesz przeniesiony na stronę logowania. W polu nazwy użytkownika wpisz **admin**, a w polu hasła zdefiniowane przed chwilą hasło. Po zalogowaniu zostaniesz poproszony o wskazanie środowiska Dockera, którym chcesz zarządzać. Masz do wyboru dwie opcje:

- Zarządzaj środowiskiem Dockera, na którym uruchomiono narzędzie Portainer (*Manage the Docker instance where Portainer is running*).
- Zarządzaj zdalnym środowiskiem Dockera (*Manage a remote Docker instance*).

Na razie chcemy zarządzać środowiskiem, w którym uruchomiono narzędzie Portainer:



Po wybraniu tej opcji na ekranie pojawia się komunikat o następującej treści:

Funkcja ta nie jest jeszcze dostępna w przypadku natywnych kontenerów Docker w systemie Windows. W systemie Linux, a także w przypadku środowiska Docker for Mac, Docker for Windows lub Docker Toolbox należy upewnić się, że kontener Portainer został uruchomiony przy użyciu flagi -v "/var/run/docker.sock:/var/run/docker.sock".

Zamontowaliśmy plik *docker.sock* podczas uruchamiania kontenera Portainer, a więc możemy zakończyć instalację, klikając przycisk *Connect* (polacz). Zostaniemy przeniesieni od razu do panelu głównego narzędzia Portainer.

Korzystanie z narzędzia Portainer

Uruchomiliśmy już narzędzie Portainer i skonfigurowaliśmy je tak, aby komunikowało się z naszym środowiskiem Docker. W związku z tym możemy przejść do funkcji wymienionych w menu znajdującym się po lewej stronie panelu głównego, który jest otwierany domyślnie po zalogowaniu się do narzędzia Portainer.

Panel główny

W panelu głównym (*Dashboard*) wyświetlane są informacje na temat instancji Dockera, z którą komunikuje się narzędzie Portainer:

The screenshot shows the Portainer dashboard interface. On the left is a sidebar with navigation links: ACTIVE ENDPOINT (local), ENDPOINT ACTIONS (Dashboard, App Templates, Containers, Images, Networks, Volumes, Events, Docker), and PORTAINER SETTINGS (Password, Users). The main area has a header "Home Dashboard" and a top right corner showing "admin log out". Below the header is a "Node info" section with a table:

| | |
|----------------|------------|
| Name | moby |
| Docker version | 17.03.1-ce |
| CPU | 2 |
| Memory | 2.1 GB |

Below this are four cards showing resource counts:

- Containers:** 1 running, 0 stopped
- Images:** 4
- Volumes:** 1
- Networks:** 3

At the bottom left of the main area is the text "Portainer v1.12.4".

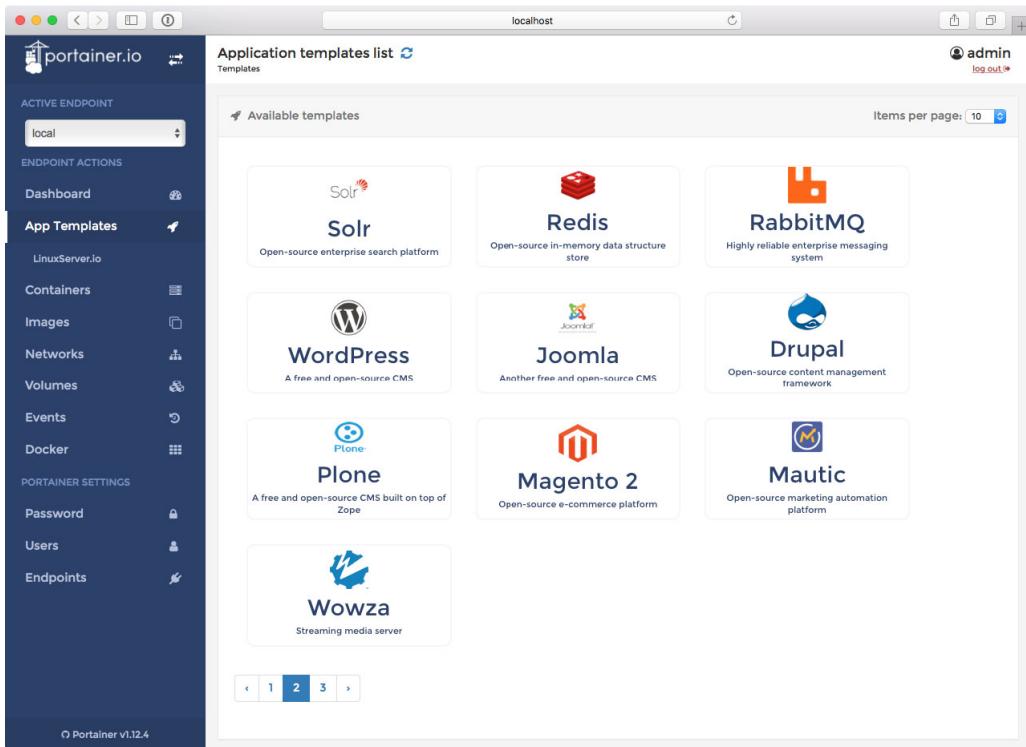
Wśród wyświetlanych informacji wymienione są uruchomione kontenery i pobrane obrazy. W moim przypadku uruchomiłem tylko jeden kontener — kontener narzędzia Portainer. Ponadto Portainer podaje informacje o wolumenach (*Volumes*) i sieciach (*Network*) dostępnych w środowisku Dockera.

Wyświetlane są też dane dotyczące samej instancji Dockera. Jak widzisz, moja instancja działa na bazie środowiska Moby Linux, dwóch procesorów i dysponuje 2 GB pamięci RAM — to domyślna konfiguracja Dockera dla systemu macOS.

Wygląd panelu głównego dostosowuje się do środowiska, w którym uruchomione jest narzędzie Portainer, a więc zajrzymy do niego jeszcze, gdy zajmiemy się dodawaniem narzędzia Portainer do klastra Docker Swarm.

Szablony aplikacji

Teraz zajmiemy się sekcją *App Templates* (szablony aplikacji). Funkcja tej sekcji jest prawdopodobnie jedyną funkcją, która nie jest dostępna bezpośrednio w jądrze Docker Engine. Umożliwia ona uruchamianie popularnych aplikacji za pomocą kontenerów pobranych z repozytorium Docker Hub:



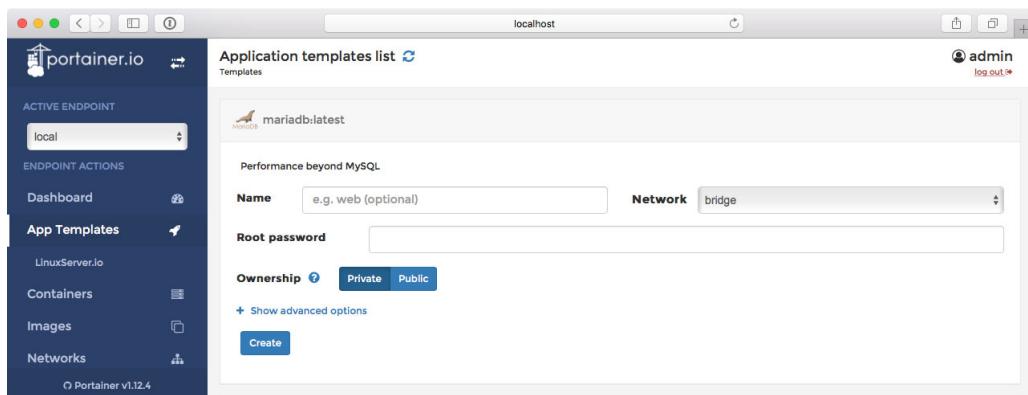
Narzędzie Portainer obsługuje 25 dodatkowych standardowo szablonów, a także szablony ze strony <https://www.linuxserver.io>.

Szablony są definiowane w formacie JSON. Oto przykład szablonu MariaDB:

```
{  
    "title": "MariaDB",  
    "description": "Performance beyond MySQL",  
    "logo": "https://cloudinovasi.id/assets/img/logos/mariadb.png",  
    "image": "mariadb:latest",  
    "env": [  
        {  
            "name": "MYSQL_ROOT_PASSWORD",  
            "label": "Root password"  
        }  
    ],  
    "ports": [  
        "3306/tcp"  
    ],  
    "volumes": ["/var/lib/mysql"]  
}
```

Kod ten wygląda podobnie jak zawartość pliku Docker Compose, ale format ten jest używany tylko przez narzędzie Portainer. Działanie większości opcji szablonów aplikacji jest jasne, ale chcielibyśmy skupić się na opcjach *Name* (nazwa) i *Label* (etykieta).

Opcje *Name* i *Label* pozwalają na wygenerowanie formularza, który będzie musiał zostać wypełniony przez użytkownika przed uruchomieniem formularza (zabieg ten przydaje się w przypadku kontenerów wymagających definiowania ustawień poprzez przekazywanie parametrów za pomocą zmiennych środowiskowych):



Jak widzisz, w przypadku kontenera MariaDB możliwe jest wybranie hasła administracyjnego. Wypełnienie pola *Root password* spowoduje przekazanie wprowadzonych wartości do uruchomionego kontenera w postaci zmiennej środowiskowej, a więc proces budowania kontenera zostanie uruchomiony za pomocą następującego polecenia:

```
$ docker container run --name [nazwa kontenera] -p 3306 -e
  ↵MYSQL_ROOT_PASSWORD=[hasło administratora] -d mariadb:latest
```

Więcej informacji na temat arkuszy aplikacji znajdziesz w dokumentacji znajdującej się na stronie <http://portainer.readthedocs.io/en/latest/templates.html>.

Kontenery

Kolejna opcja menu znajdującej się po lewej stronie ekranu to kontenery (*Containers*). Pozwala ona na uruchamianie kontenerów Dockera i nawiązywanie z nimi interakcji. Po kliknięciu przycisku *Containers* zobaczysz listę wszystkich kontenerów (aktywnych i zatrzymywanych) obecnych w środowisku Dockera (patrz pierwszy zrzut na następnej stronie).

Jak widzisz, w moim środowisku aktywny jest tylko jeden kontener (kontener aplikacji Portainer). Spróbujmy uruchomić aplikację działającą w klastrze, z której korzystaliśmy w poprzednich rozdziałach. Kliknij przycisk *+Add Container* (dodaj kontener).

The screenshot shows the Portainer.io interface for managing Docker containers. On the left is a sidebar with various navigation options like Dashboard, App Templates, Containers, Images, Networks, Volumes, Events, and Docker. The main area is titled 'Container list' and shows a table of containers. A single row is selected, labeled 'running' and named 'jovial_hodgkin'. The table includes columns for State, Name, Image, IP Address, Published Ports, and Ownership. At the top of the table are buttons for Start, Stop, Kill, Restart, Pause, Resume, Remove, Add container, and Show all containers.

Na stronie *Create Container* (utwórz kontener) należy zdefiniować następujące parametry:

- *Name* (nazwa): cluster,
- *Image* (obraz): russmckendrick/cluster,
- *Always pull the image* (zawsze pobieraj obraz): włącz tę funkcję,
- *Publish all exposed ports* (publikuj wszystkie odsłonięte porty): włącz tę funkcję.

Na koniec określ mapowanie portu — port 80 kontenera połącz z portem 80 hosta, klikając przycisk *+map additional port* (mapuj dodatkowy port). Formularz gotowy do zatwierdzenia powinien wyglądać tak:

The screenshot shows the 'Create container' form in Portainer.io. The 'Name' field is set to 'cluster'. The 'Image' field is set to 'russmckendrick/cluster'. The 'Always pull the image' toggle switch is turned on. In the 'Ports configuration' section, the 'Publish all exposed ports' toggle switch is turned on. Under 'Port mapping', there is a table with one entry: 'host' port 80 mapped to 'container' port 80. The 'TCP' button is selected. The 'Ownership' section shows 'Private' selected. At the bottom are 'Start container' and 'Cancel' buttons.

Po zakończeniu wprowadzania danych kliknij przycisk *Start container* (uruchom kontener). Po kilku sekundach na ekranie pojawi się lista aktywnych kontenerów. Powinieneś znaleźć na niej uruchomiony przed chwilą kontener:

The screenshot shows the Portainer interface with the title 'Container list'. On the left sidebar, under 'ACTIVE ENDPOINT', 'local' is selected. Under 'CONTAINERS', 'Containers' is selected. The main area displays a table of containers with the following data:

| State | Name | Image | IP Address | Published Ports | Ownership |
|---------|----------------|-------------------------------|------------|-----------------|---|
| running | cluster | russmckendrick/cluster:latest | 172.17.0.3 | 80:80 | <input checked="" type="checkbox"/> Private <input type="checkbox"/> Switch to public |
| running | jovial_hodgkin | portainer/portainer | 172.17.0.2 | 9000:9000 | <input type="checkbox"/> Public |

Pola wyboru znajdujące się po lewej stronie listy kontenerów aktywują górny ciąg przycisków umożliwiających sterowanie statusem kontenerów — nie zamkaj (*Kill*) ani nie usuwaj (*Remove*) kontenera Portainer. Po kliknięciu nazwy kontenera wyświetlane zostaną szczegółowe informacje o wybranym kontenerze. Oto przykład informacji opisujących kontener *cluster*:

The screenshot shows the Portainer interface with the title 'Container details'. On the left sidebar, under 'ACTIVE ENDPOINT', 'local' is selected. Under 'CONTAINERS', 'Containers' is selected. The main area displays the details for the 'cluster' container:

- Actions:** Buttons for Start, Stop, Kill, Restart, Pause, Resume, and Remove.
- Container status:**
 - Name: cluster
 - IP address: 172.17.0.3
 - Status: Running since 3 minutes
 - Start time: 2017-05-07 14:26:52
- Create image:**
 - Description: You can create an image from this container, this allows you to backup important data or save helpful configurations. You'll be able to spin up another container based on this image afterward.
 - Form fields:
 - Name: e.g. myImage:myTag
 - Registry: optional
 - Note: If you don't specify the tag in the image name, `latest` will be used.
 - Create button
- Container details:**
 - Image: sha256:0b2af6cff8336f2d59b0e5ab550aa3152cb33f0c24a483e5cf2ec3eb008f4dd9
 - Port configuration: 80/tcp → 0.0.0.0:80
 - CMD: -c /etc/supervisord.conf
 - ENV: PATH /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

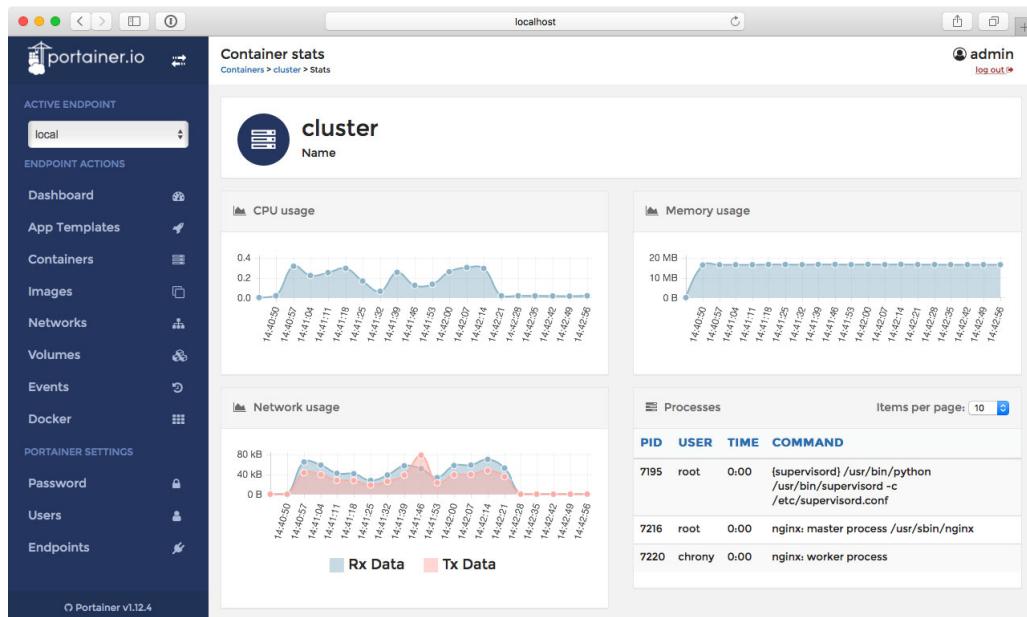
Jak widzisz, są to te same informacje, które można uzyskać, wywołując polecenie:

```
$ docker container inspect cluster
```

Możesz również skorzystać z przycisków *Stats* (statystyki), *Logs* (dzienniki) i *Console* (konsola).

Statystyki

W sekcji *Stats* (statystyki) znajdziesz informacje o obciążeniu procesora, pamięci i połączenia sieciowego. Dodatkowo przedstawiono tutaj listę procesów uruchomionych w wybranym kontenerze:



Jeżeli pozostawisz tę stronę otwartą, to wykresy będą automatycznie aktualizowane. Odświeżenie strony spowoduje rozpoczęcie procesu generowania wykresów od początku. Dzieje się tak, ponieważ narzędzie Portainer otrzymuje te dane z interfejsu programistycznego Dockera za pomocą polecenia:

```
$ docker container stats cluster
```

Każde odświeżenie strony powoduje wywołanie tego polecenia od początku, ponieważ Portainer nie komunikuje się z Dockerem w tle w celu zapisywania statystyk wszystkich aktywnych kontenerów.

Dzienniki

Sekcja *Logs* (dzienniki) pokazuje wyniki wywołania następującego polecenia:

```
$ docker container logs cluster
```

Portainer wyświetla dzienniki STDOUT i STDERR:

```

2017-05-07 13:26:53,010 CRIT Set uid to user 0
2017-05-07 13:26:53,019 INFO RPC interface 'supervisor' initialized
2017-05-07 13:26:53,019 CRIT Server 'unix_http_server' running without any HTTP authentication checking
2017-05-07 13:26:53,019 INFO supervisord started with pid 1
2017-05-07 13:26:54,023 INFO spawned: 'nginx' with pid 8
2017-05-07 13:26:54,026 INFO spawned: 'start' with pid 9
2017-05-07 13:26:54,032 INFO exited: start (exit status 0; not expected)
2017-05-07 13:26:54,032 INFO gave up: start entered FATAL state, too many start retries too quickly
2017-05-07 13:26:55,033 INFO success: nginx entered RUNNING state, process has stayed up for > than 1 seconds (startsecs )
)

```

Do wyświetlanych danych można również dodać znaczniki czasu. Opcja ta jest odpowiednikiem polecenia:

```
$ docker container logs --timestamps cluster
```

Konsola

Ostatnia sekcja *Console* (konsola) umożliwia zalogowanie się do kontenera za pomocą Terminala napisanego w HTML5. Przed nawiązaniem połączenia z kontenerem musisz wybrać powłokę. Możesz skorzystać z dwóch powłok: */bin/bash* lub */bin/sh*. W obrazie *cluster* zainstalowano obie powłoki, a więc zdecydowałem się na wybór powłoki *sh* (patrz pierwszy zrzut na następnej stronie).

Korzystanie z tej sekcji odpowiada uruchomieniu polecenia:

```
docker container exec -it cluster /bin/sh
```

Z powyższego zrzutu wynika, że procesowi *sh* przypisano identyfikator procesu (*PID*) 13. Proces ten został utworzony przez polecenie *docker container exec*. Zostanie on zakończony po odłączeniu od sesji powłoki.

Obrazy

Kolejną opcję menu znajdującej się po lewej stronie ekranu jest *Images* (obrazy). Opcja ta umożliwia zarządzanie obrazami, pobieranie ich i wysyłanie (patrz drugi zrzut na następnej stronie).

Docker. Programowanie aplikacji dla zaawansowanych

The screenshot shows the Portainer.io interface for managing Docker containers. On the left is a sidebar with navigation links for Dashboard, App Templates, Containers, Images, Networks, Volumes, Events, Docker, and Portainer Settings. The 'Containers' link is highlighted. The main area is titled 'Container console' and shows a terminal session connected to a container named 'alpine'. The terminal output displays system information like /etc/*release, a ps aux command showing processes like supervisord, nginx master and worker processes, and a free -m command showing memory usage. It also shows the contents of the /var/www/html directory, which includes index.html and swarm.png files.

The screenshot shows the Portainer.io interface for managing Docker images. The sidebar has the 'Images' link highlighted. The main area is titled 'Image list' and shows a table of images. The table has columns for Id, Tags, Size, and Created. The data in the table is as follows:

| Id | Tags | Size | Created |
|----------------------|-------------------------------|---------|---------------------|
| sha256:20298e16e9... | traefik:latest | 41.8 MB | 2017-04-13 21:25:33 |
| sha256:0b2af6cff8... | russmckendrick/cluster:latest | 51.4 MB | 2017-03-03 20:39:13 |
| sha256:89883cee36... | portainer/portainer:latest | 10 MB | 2017-04-06 09:41:40 |
| sha256:4a15e3663... | alpine:latest | 4 MB | 2017-03-03 20:32:37 |

W górnej części strony znajduje się pole pobierania obrazów. Wprowadzenie w nim nazwy kontenera takiego jak np. *amazonlinux* i kliknięcie przycisku *Pull* (pobierz) spowoduje pobranie kopii obrazu kontenera Amazon Linux z repozytorium Docker Hub. W takim przypadku Portainer uruchomi w tle polecenie:

```
$ docker image pull amazonlinux
```

Kliknięcie identyfikatora obrazu spowoduje wyświetlenie szczegółowych informacji opisujących wybrany obraz. Wyświetlona strona w czytelny sposób prezentuje dane zwracane przez polecenie:

```
$ docker image inspect russmckendrick/cluster
```

Przyjrzyj się poniższemu zrzutowi:

The screenshot shows the Portainer web application interface. On the left is a sidebar with navigation links: ACTIVE ENDPOINT (local), ENDPOINT ACTIONS (Dashboard, App Templates, Containers, Images, Networks, Volumes, Events, Docker), and PORTAINER SETTINGS (Password, Users, Endpoints). The main content area is titled "Image details" for the image ID `sha256:0b2af6cff8336f2d59b0e5ab550aa3152cb33f0c24a483e5cf2ec3eb008f4dd9`. It includes sections for "Image tags" (listing `russmckendrick/cluster:latest`), "Tag the image" (with fields for Name and Registry), "Image details" (showing ID, Size, Created, Build, Author), and "Dockerfile details" (listing CMD, ENTRYPOINT, EXPOSE, ENV). A red "Delete this image" button is visible next to the ID.

Poza informacjami o obrazie na ekranie znajdują się również przyciski umożliwiające załadowanie kopii obrazu do wybranego repozytorium (domyślnie jest to Docker Hub).

Sieci i wolumeny

Kolejne dwie opcje menu (*Networks* i *Volumes*) pozwalają na zarządzanie sieciami i wolumenami. Nie będę ich opisywał w sposób szczegółowy, ponieważ korzystanie z nich jest proste.

Sieci

W sekcji *Networks* (sieci) możesz sprawnie dodać sieć, korzystając z domyślnego sterownika mostka. Kliknięcie przycisku *Advanced settings* (ustawienia zaawansowane) spowoduje otwarcie strony z dodatkowymi opcjami. Wśród nich znajdziesz możliwość skorzystania z innych sterowników, zdefiniowania podsieci, dodania etykiet i ograniczenia dostępu do sieci z zewnątrz. Ponadto możesz usuwać utworzone wcześniej sieci i sprawdzać ich konfigurację.

Wolumeny

W sekcji *Volumes* (wolumeny) możesz w zasadzie tylko dodawać i usuwać wolumeny. Dodając wolumen, możesz wybrać sterownik, a także zdefiniować parametry przekazywane do sterownika, co pozwala na korzystanie z wtyczek niezależnych sterowników. W sekcji tej nie ma zbyt wielu opcji. Nie ma nawet możliwości wyświetlenia konfiguracji utworzonych wcześniej wolumenów.

Zdarzenia

W sekcji *Events* (zdarzenia) znajdziesz informacje o wszystkich zdarzeniach, do których doszło w ciągu ostatnich 24 godzin. Ponadto możesz filtrować wyświetlane dane, co pozwala na szybkie znalezienie potrzebnych informacji:

| Date | Category | Details |
|---------------------|-----------|---------------------------------|
| 2017-05-07 15:53:17 | image | Image amazonlinux:latest pulled |
| 2017-05-07 15:13:28 | container | Exec instance started |
| 2017-05-07 15:13:28 | container | Exec instance created |
| 2017-05-07 15:13:24 | container | Exec instance started |
| 2017-05-07 15:13:23 | container | Exec instance created |
| 2017-05-07 15:12:46 | container | Exec instance started |
| 2017-05-07 15:12:46 | container | Exec instance created |
| 2017-05-07 15:12:33 | container | Exec instance started |
| 2017-05-07 15:12:33 | container | Exec instance created |
| 2017-05-07 15:12:27 | container | Exec instance started |

Dane wyświetlane w tej sekcji odpowiadają danym zwracanym w wyniku wywołania polecenia:

```
$ docker events --since '2017-05-06T16:30:00' --until '2017-05-07T16:30:00'
```

Docker

W sekcji *Docker* znajdziesz informacje wygenerowane w wyniku wywołania polecenia:

```
$ docker info
```

Oto zrzut prezentujący takie informacje:

The screenshot shows the Portainer interface for Docker. On the left is a sidebar with a dark blue background and white text, listing various sections: ACTIVE ENDPOINT (local), ENDPOINT ACTIONS (Dashboard, App Templates, Containers, Images, Networks, Volumes, Events), Docker (selected), and PORTAINER SETTINGS (Password, Users, Endpoints). The Docker section has a grid icon. At the bottom of the sidebar, it says "Portainer v1.12.4". The main content area has a light gray background and is titled "Engine overview". It contains three sections: "Engine version", "Engine status", and "Engine plugins".

| Engine version | |
|----------------|-------------------|
| Version | 17.03.1-ce |
| API version | 1.27 |
| Go version | go1.7.5 |
| OS type | linux |
| OS | Alpine Linux v3.5 |
| Architecture | amd64 |
| Kernel version | 4.9.13-moby |

| Engine status | |
|-----------------------|-----------------|
| Total CPU | 2 |
| Total memory | 2.1 GB |
| Docker root directory | /var/lib/docker |
| Storage driver | overlay2 |
| Logging driver | json-file |
| Cgroup driver | cgroups |

| Engine plugins | |
|----------------|--|
| Volume | local |
| Network | bridge, host, ipvlan, macvlan, null, overlay |

Korzystanie z tej sekcji przydaje się w przypadku pracy z wieloma węzłami końcowymi Dockera i potrzeby uzyskania informacji na temat środowiska wybranego węzła końcowego.

Portainer i Docker Swarm

W poprzednim podrozdziale opisywaliśmy korzystanie z narzędzia Portainer w samodzielnej instancji Dockera. Portainer obsługuje również klastry Docker Swarm. Interfejs tego narzędzia pozwala na przystosowanie go do środowiska klastrów. W tym podrozdziale przyjrzymy się zagadnieniom związanym z uruchamianiem klastra Swarm i narzędzia Portainer w charakterze usługi.

Tworzenie klastra

Podobnie jak w poprzednim rozdziale utworzymy lokalny klaster Swarm oparty na maszynie Docker Machine. W tym celu musimy uruchomić następujące polecenia:

```
$ docker-machine create \
-d virtualbox \
swarm-manager
```

```
$ docker-machine create \
-d virtualbox \
swarm-worker01
```

```
$ docker-machine create \
-d virtualbox \
swarm-worker02
```

Po uruchomieniu trzech instancji musimy uruchomić polecenie inicjalizujące menedżera Swarm:

```
$ docker $(docker-machine config swarm-manager) swarm init \
--advertise-addr $(docker-machine ip swarm-manager):2377 \
--listen-addr $(docker-machine ip swarm-manager):2377
```

Czas na uruchomienie poleceń dodających węzły robocze (pamiętaj o korzystaniu z własnego kodu dostępowego):

```
$ docker $(docker-machine config swarm-worker01) swarm join \
$(docker-machine ip swarm-manager):2377 \
--token SWMTKN-1-5wz1e5n1c2oqlaojbbr1mlwvc3wjrh8iefvs889d1uzc5r8ag1-
→9rwp0n4q37jsw5fwh30xn4h1d
```

```
$ docker $(docker-machine config swarm-worker02) swarm join \
$(docker-machine ip swarm-manager):2377 \
--token SWMTKN-1-5wz1e5n1c2oqlaojbbr1mlwvc3wjrh8iefvs889d1uzc5r8ag1-
→9rwp0n4q37jsw5fwh30xn4h1d
```

Po utworzeniu klastra należy uruchomić poniższe polecenie w celu wskazania lokalnemu klientowi Dockera węzła menedżera:

```
$ eval $(docker-machine env swarm-manager)
```

Na koniec możemy sprawdzić status klastra:

```
$ docker node ls
```

Usługa Portainer

Utworzyliśmy kластер Docker Swarm i skonfigurowaliśmy lokalnego klienta Dockera tak, aby komunikował się z menedżerem węzła, a więc możemy uruchomić usługę Portainer:

```
docker service create \
  --name portainer \
  --publish 9000:9000 \
  --constraint 'node.role == manager' \
  --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
  portainer/portainer \
  -H unix:///var/run/docker.sock
```

Polecenie to uruchomi narzędzie Portainer jako usługę działającą w węźle menedżera i sprawi, że usługa ta zamontuje plik docker.sock zapewniający widoczność pozostałych węzłów klastra. Poprawność uruchomienia usługi można sprawdzić za pomocą polecień:

```
$ docker service ls
$ docker service inspect portainer --pretty
```

Oto informacje wyświetcone po uruchomieniu drugiego z wymienionych polecień:

```
russ in ~
2. russ (bash) 🎙
⚡ docker service inspect portainer --pretty

ID:          1kcayhodyz6onbdj8l1103b7v
Name:        portainer
Service Mode: Replicated
Replicas:    1
Placement:Constraints: [node.role == manager]
UpdateConfig:
  Parallelism: 1
  On failure:  pause
  Max failure ratio: 0
ContainerSpec:
  Image:        portainer/portainer:latest@sha256:40bf7e42c9cd4b95ab70b9eca8c9b772e7ef65e78fa094ccb6f745e117b5237c
  Args:         -H unix:///var/run/docker.sock
Mounts:
  Target = /var/run/docker.sock
  Source = /var/run/docker.sock
  ReadOnly = false
  Type = bind
Resources:
Endpoint Mode: vip
Ports:
  PublishedPort 9000
  Protocol = tcp
  TargetPort = 9000
russ in ~
⚡
```

Po uruchomieniu usługi możesz uzyskać dostęp do narzędzia Portainer na porcie nr 9000 i pod adresem IP dowolnego węzła wchodzącego w skład klastra. Możesz również skorzystać z polecenia:

```
$ open http://$(docker-machine ip swarm-manager):9000
```

Po otwarciu strony w przeglądarce zostaniesz ponownie poproszony o zdefiniowanie hasła administratora, a następnie otworzy się strona logowania. Po zalogowaniu się zobaczysz główny panel narzędzia Portainer. Dzieje się tak, ponieważ tym razem podczas uruchamiania narzędzia Portainer przekazaliśmy argument `-H unix:///var/run/docker.sock` i nie musimy ręcznie definiować ustawień, które konfigurowaliśmy podczas uruchamiania narzędzia Portainer na jednym lokalnym hoście.

Różnice związane z pracą w klastrze

Interfejs narzędzia Portainer połączonego z klastrem Docker Swarm wygląda nieco inaczej. W tej sekcji zajmiemy się opisem tych różnic. Jeżeli jakaś część interfejsu narzędzia Portainer nie została opisana w tej sekcji, to znaczy, że wygląda ona tak samo jak w przypadku pracy z pojedynczym hostem.

Panel główny

Jedną z najbardziej widocznych różnic jest to, że teraz w panelu głównym wyświetlane są informacje o klastrze Swarm:

The screenshot shows the Portainer.io main dashboard for a Swarm cluster. On the left, there's a sidebar with navigation links: ACTIVE ENDPOINT (primary), ENDPOINT ACTIONS (Dashboard, App Templates, Services, Containers, Images, Networks, Volumes, Swarm), and PORTAINER SETTINGS (Password, Users, Endpoints). The main content area is titled 'Home Dashboard'. It displays 'Node info' for the node 'swarm-manager' (Docker version 17.05.0-ce, CPU 1, Memory 1 GB) and 'Swarm info' (This node is part of a Swarm cluster, Node role Manager, Nodes in the cluster 3). Below this, there are four summary cards: 1 Container (1 running, 0 stopped), 1 Image (10 MB), 1 Volume (aufs driver), and 5 Networks.

W panelu głównym znajduje się teraz mała sekcja przedstawiająca podstawowe informacje na temat rozmiaru klastra, a także roli węzła Docker Swarm, z którym aktualnie komunikuje się narzędzie Portainer.

Klaster Swarm

Po kliknięciu przycisku *Swarm* znajdującego się w menu widocznym po lewej stronie ekranu wyświetlane zostaną szczegółowe informacje na temat klastra, a także statusu poszczególnych węzłów:

The screenshot shows the 'Cluster overview' section of the Portainer interface. It displays the following information:

- Nodes:** 3
- Docker API version:** 1.29
- Total CPU:** 3
- Total memory:** 3.13 GB

Below this, the 'Node status' table lists three nodes:

| Name | Role | CPU | Memory | Engine | IP Address | Status |
|----------------|---------|-----|--------|------------|----------------|--------|
| swarm-worker01 | worker | 1 | 1 GB | 17.05.0-ce | 192.168.99.101 | ready |
| swarm-manager | manager | 1 | 1 GB | 17.05.0-ce | 192.168.99.100 | ready |
| swarm-worker02 | worker | 1 | 1 GB | 17.05.0-ce | 192.168.99.102 | ready |

Kliknięcie nazwy węzła spowoduje otwarcie strony zawierającej więcej opcji konfiguracji wybranego węzła. Na stronie tej możesz również określić dostępność węzła w klastrze, a także go wstrzymać lub rozpoczęć proces jego drenowania:

The screenshot shows the 'Node details' section for the 'swarm-worker01' node. The configuration includes:

- Name:** my-mana (input field)
- Host name:** swarm-worker01
- Role:** worker
- Availability:** Active (dropdown)
- Status:** ready (button)

Below this, there is a note about Docker Swarm mode Node documentation and an 'Apply changes' button.

The 'Node description' section contains the following hardware specifications:

| | |
|-----------------------|--------------|
| CPU | 1 |
| Memory | 1.04 GB |
| Platform | linux x86_64 |
| Docker Engine version | 17.05.0-ce |

The 'Node labels' section indicates: 'There are no labels for this node.'

Usługi

Strona *Services* (usługi) umożliwia tworzenie usług i zarządzanie nimi. Obecnie powinna na niej znajdować się tylko jedna usługa (Portainer). Nie chcemy zakłócać pracy kontenera usługi Portainer, a więc utworzymy nową usługę. W tym celu kliknijmy przycisk **+ Add Service** (dodaj usługę) i na stronie, która zostanie załadowana, wprowadźmy następujące dane:

- *Name* (nazwa): cluster,
- *Image* (obraz): russmckendrick/cluster,
- *Scheduling mode* (tryb harmonogramu): *Replicated* (powtarzany),
- *Replicas* (liczba replik): 1.

Ponownie dodaj mapowanie portu 80 hosta z portem 80 kontenera:

Po wprowadzeniu wszystkich ustawień kliknij przycisk *Create service* (utwórz usługę). Na ekranie pojawi się lista usług, która powinna już zawierać dodaną przed chwilą usługę o nazwie *cluster*:

| Name | Image | Scheduling mode | Published Ports | Updated at | Ownership |
|-----------|-------------------------------|------------------------|-----------------|---------------------|--------------------------|
| cluster | russmckendrick/cluster:latest | replicated 1 / 1 Scale | 80:80 | 2017-05-07 18:37:33 | Private Switch to public |
| portainer | portainer/portainer:latest | replicated 1 / 1 Scale | 9000:9000 | 2017-05-07 18:07:21 | Public |

W kolumnie *Scheduling mode* znajduje się przycisk *Scale* (skaluj). Kliknijmy go i zwiększymy liczbę replik *cluster* do 6.

Kliknięcie nazwy *cluster* znajdującej się w kolumnie *Name* (nazwa) spowoduje wyświetlenie informacji o tej usłudze. Są one dość obszerne:

The screenshot shows the Portainer interface with the URL 192.168.99.100. On the left, the sidebar has 'primary' selected under 'ACTIVE ENDPOINT'. Under 'ENDPOINT ACTIONS', 'Services' is selected. The main area shows 'Service details' for the 'cluster' service. The 'Replicas' field is highlighted and set to 6. Other details include: ID: ru64klxxgx5x5s2v0e8e7s47e, Created at: 2017-05-07 18:37:33, Last updated at: 2017-05-07 18:46:21, Version: 36, Scheduling mode: replicated, and Image: russmckendrick/cluster:latest@sha256:e7d6182aa203671889d940a12918278b. A sidebar on the right lists various configuration options like Environment variables, Container labels, Mounts, etc.

Menu *Service* umożliwia zmianę wielu opcji usług „w locie”. Dotyczy to między innymi ograniczeń umiejscawiania usługi, zasady ponownego uruchamiania i etykiety. W dolnej części strony znajduje się lista zadań związanych z usługą:

The screenshot shows the Portainer interface with the URL 192.168.99.100. On the left, the sidebar has 'primary' selected under 'ACTIVE ENDPOINT'. Under 'ENDPOINT ACTIONS', 'Services' is selected. The main area shows 'Associated tasks' for the 'cluster' service. There are six tasks listed, all in 'running' status: 2ffn9owf49428299bzthzx5t2, jzes7le4soqhfftdd4dunloa8f, l6wvhdimuyg8dfhqx398vmc96, alih7elj0qldg08ztrq5u79w, Sqbzv045hyhnu6ek5zq47l0wg2, and 9sjn6qzoujgahp3wx14yuvtkp. The tasks are listed in a table with columns: Id, Status, Slot, Node, and Last update. The last update column shows dates from May 7, 2017, at 18:37:42 to 18:46:29.

Jak widzisz, mamy sześć aktywnych zadań — po dwa w każdym z trzech węzłów. Kliknięcie przycisku *Containers* (kontenery) znajdującego się w menu umieszczonego po lewej stronie ekranu spowoduje wyświetlenie treści innej, niż mógłbyś się spodziewać:

The screenshot shows the Portainer.io web interface. On the left is a sidebar with navigation links: ACTIVE ENDPOINT (primary), ENDPOINT ACTIONS (Dashboard, App Templates, Services, Containers, Images), and footer text "O Portainer v1.12.4". The main area is titled "Container list" and shows a table of "Containers". The table has columns: State, Name, Image, IP Address, Published Ports, and Ownership. There are three entries, all in the "running" state:

| State | Name | Image | IP Address | Published Ports | Ownership |
|---------|--------------------------------------|-------------------------------|-------------|-----------------|-----------------|
| running | cluster.4.alhi7e1j0q1dg08ztrq5u79w | russmckendrick/cluster:latest | 10.255.0.11 | - | Private service |
| running | cluster.2.jzes7le4soqhftd4dunloa8f | russmckendrick/cluster:latest | 10.255.0.9 | - | Private service |
| running | portainer.1.m7yfu7ao57zsdamz3rp9f4ip | portainer/portainer:latest | 10.255.0.6 | - | Public service |

Na liście znajdują się tylko trzy kontenery i jednym z nich jest usługa Portainer. Dlaczego tak się dzieje?

Jak zapewne pamiętasz z lektury poprzedniego rozdziału, polecenia docker container są wykonywane w kontekście tylko wybranego węzła, a narzędzie Portainer komunikuje się tylko z węzłem będącym menedżerem i to w kontekście właśnie tego węzła wykonywane są polecenia docker container. Pamiętaj o tym, że narzędzie Portainer to tylko interfejs graficzny korzystający z interfejsu programistycznego Dockera, a więc wyświetla on te same informacje, które są wyświetlane w wyniku uruchomienia w trybie tekstowym polecenia docker container ls.

Węzły końcowe

Do narzędzia Portainer możemy dodać nasze dwa pozostałe klastry. W tym celu należy skorzystać z opcji *Endpoints* (węzły końcowe) znajdującej się w menu wyświetlonym po lewej stronie ekranu.

W celu dodania węzła końcowego musimy znać jego adres URL i mieć dostęp do certyfikatów umożliwiających uwierzytelnienie narzędzia Portainer w demonie Dockera sterującym pracą wybranego węzła. Na szczęście nasze hosty zostały uruchomione za pomocą narzędzia Docker Machine, a więc zadanie to nie jest trudne. W celu uzyskania adresów URL węzłów końcowych wystarczy uruchomić polecenie:

```
$ docker-machine ls
```

W przypadku mojego klastra węzłom końcowym przypisano adresy URL 192.168.99.101:2376 i 192.168.99.102:2376 (w przypadku Twojego klastra adresy mogą być inne). Certyfikaty, których potrzebujemy, znajdują się w folderze `~/docker/machine/certs/`. Polecam otwieranie tego folderu za pomocą następujących polecień:

```
$ cd ~/docker/machine/certs/  
$ open .
```

Po dodaniu węzła można do niego przejść za pomocą rozwijanego menu *Active Endpoint* (aktywny węzeł końcowy) znajdującego się w górnym lewym rogu strony. Ze znajdującego się niżej zrzutu wynika, że w węźle *swarm-worker-01* uruchomione są dwa kontenery, a w klastrze odgrywa on rolę węzła roboczego:

The screenshot shows the Portainer interface at the URL 192.168.99.100. The sidebar on the left lists various sections: ACTIVE ENDPOINT (set to 'swarm-worker01'), ENDPOINT ACTIONS, Dashboard, App Templates, Services, Containers, Images, Networks, Volumes, and PORTAINER SETTINGS (Password, Users, Endpoints). The main content area is titled 'Home' and 'Dashboard'. It shows 'Node info' for 'swarm-worker01' with Docker version 17.05.0-ce, 1 CPU, and 1 GB of memory. Below that is 'Swarm info' stating 'This node is part of a Swarm cluster' and 'Node role: Worker'. At the bottom, there are four summary boxes: 'Containers' (2 running, 0 stopped), 'Images' (1), 'Volumes' (0), and 'Networks' (5).

W głównym panelu poza małą sekcją *Swarm info* (informacje o klastrze Swarm) nie ma innych informacji o usługach Swarm. Wynika to z tego, że narzędzie Portainer dysponuje informacjami dostarczanymi przez węzły Docker'a, a w trybie Docker Swarm usługi mogą być uruchamiane tylko przez węzły będące menedżerami. Tylko one mogą rozdzielać zadania i nawiązywać interakcje z innymi węzłami klastra.

Podsumowanie

To koniec zgłębiania tematyki związanej z narzędziem Portainer. Jak widzisz, Portainer ma wiele funkcji, jest prosty w obsłudze i rozwija się wraz z całym ekosystemem Docker'a. Narzędzie to pozwala na wykonanie różnych operacji związanych z hostami, kontenerami i usługami uruchomionymi w pojedynczym hoście lub w klastrze.

W kolejnym rozdziale zajmiemy się aplikacją Rancher — kolejnym narzędziem będącym graficznym interfejsem służącym do zarządzania hostami Docker'a, kontenerami i obrazami.

Rancher

Rancher to kolejny otwarty projekt, który pomaga we wdrażaniu środowisk Dockera. Jest to graficzny interfejs umożliwiający wykonywanie wszystkich operacji dostępnych z poziomu wiersza poleceń, który ponadto oferuje kilka dodatkowych funkcji.

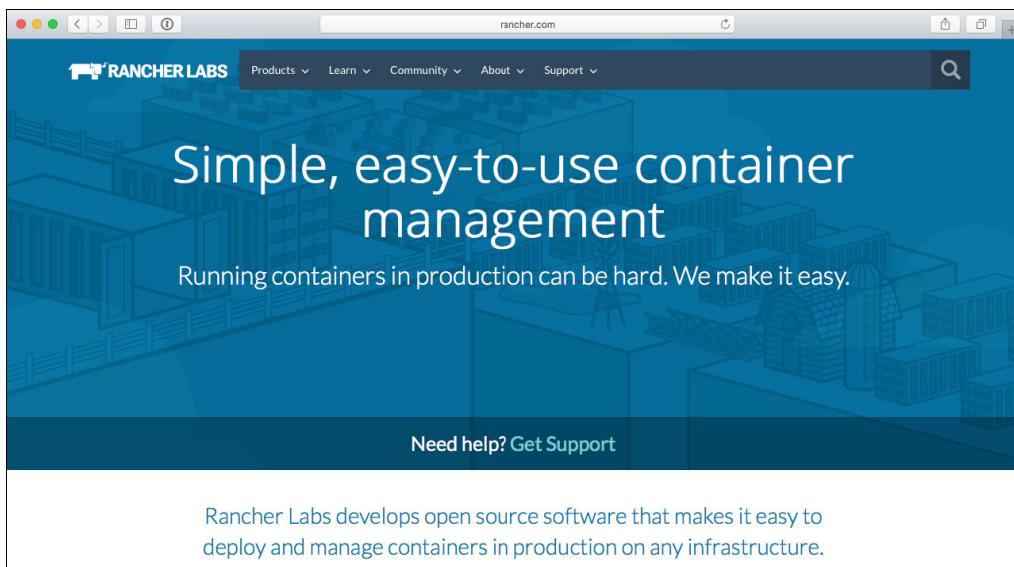
W tym rozdziale zajmiemy się następującymi zagadnieniami:

- instalacja i konfiguracja uwierzytelniania;
- tworzenie stad;
- uruchamianie stosów;
- usuwanie stad;
- inne typy klastrów.

Instalacja i konfiguracja uwierzytelniania

Strona narzędzia Rancher (<http://rancher.com/>) przypomina stronę narzędzia Portainer (patrz zrzut na następnej stronie).

Oba wspomniane narzędzia mogą być użyte do wdrażania kontenerów i zarządzania nimi, ale funkcjonują one w dwóch różnych przestrzeniach. Różnice pomiędzy nimi widać już podczas instalacji. Rancher obecnie nie obsługuje wersji Dockera przeznaczonych dla systemów macOS i Windows. Jeżeli korzystasz z tych platform, to możesz skorzystać z maszyny wirtualnej, w której zainstalujesz system Ubuntu w wersji 16.04 lub nowszej (maszyna musi dysponować przynajmniej 1 GB pamięci RAM), ale w celu wykorzystania pełnych możliwości narzędzia Rancher polecam Ci skorzystanie z jego ogólnodostępnej wersji serwerowej.



Instalacja

Podeczas instalacji będę korzystał z apletu chmury DigitalOcean i narzędzia Docker Machine. Komponenty te będą stanowiły bazę usług Rancher.

W celu uruchomienia apletu (*droplet*) chmury DigitalOcean należy uruchomić polecenie:

```
$ docker-machine create \
--driver digitalocean \
--digitalocean-access-token
7b6216feba5d02669525bf8e7a86c773c9d17a438987c08040ce32f14b66d4cf \
--digitalocean-size 1gb \
rancher
```

W poleceniu musisz umieścić własny klucz dostępowy. Możesz go wygenerować na stronie <https://cloud.digitalocean.com/settings/api>.

Po uruchomieniu apletu możemy wskazać go w lokalnym kliencie Dockera za pomocą polecenia:

```
$ eval $(docker-machine env rancher)
```

Gdy klient komunikuje się z apletem chmury, możemy uruchomić narzędzie Rancher za pomocą polecenia:

```
$ docker container run -d \
--name=rancher \
```

```
--restart=unless-stopped \
-p 8080:8080 \
rancher/server:stable
```

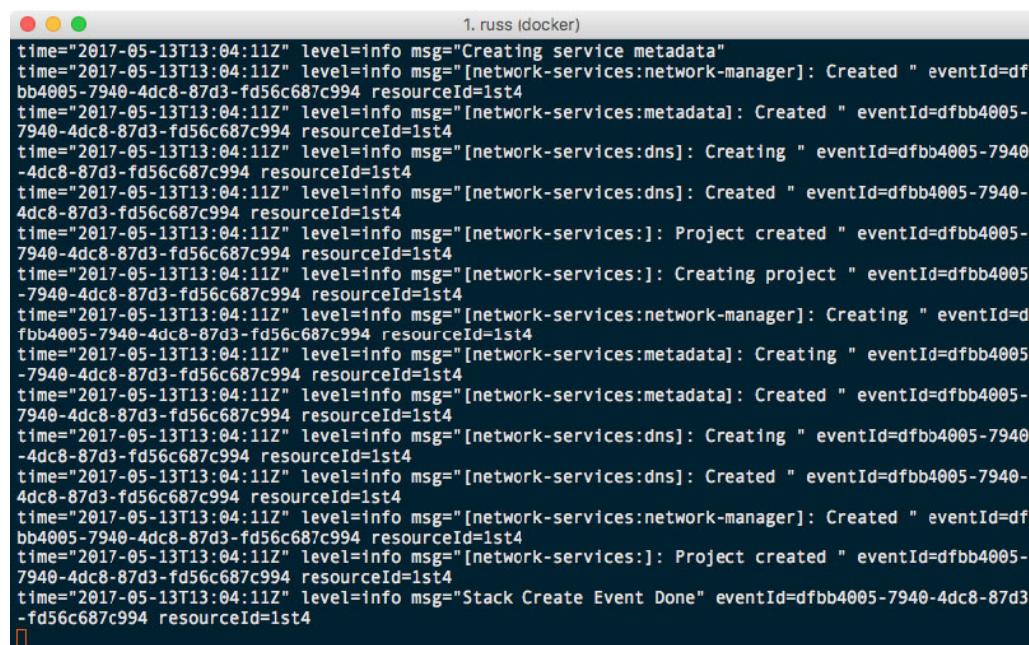
Narzędzie Rancher potrzebuje kilku minut na zainstalowanie. Postęp procesu instalacji można śledzić za pomocą polecenia:

```
docker container logs rancher -f
```

Jeżeli podczas instalowania narzędzia Rancher wyświetlane zostaną komunikaty podobne do poniższych, a kontener Rancher zostanie zatrzymany, to znaczy, że dysponuje on zbyt małą ilością pamięci RAM:

```
# There is insufficient memory for the Java Runtime Environment to continue.
# Native memory allocation (mmap) failed to map 31092736 bytes for committing
→reserved memory.
```

Sprawdź, czy podczas uruchamiania apletu chmury zastosowano flagę `--digitalocean-size 1gb`. O zakończeniu procesu instalacji świadczy niepojawianie się nowych wpisów w dzienniku. Ostatnie komunikaty powinny informować o utworzeniu poszczególnych elementów narzędzia Rancher. Oto przykład takich komunikatów:



```
1. russ (docker)
time="2017-05-13T13:04:11Z" level=info msg="Creating service metadata"
time="2017-05-13T13:04:11Z" level=info msg="[network-services:network-manager]: Created " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:metadata]: Created " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:dns]: Creating " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:dns]: Created " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:]: Project created " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:]: Creating project " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:network-manager]: Creating " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:metadata]: Creating " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:metadata]: Created " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:dns]: Creating " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:dns]: Created " eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
time="2017-05-13T13:04:11Z" level=info msg="[network-services:dns]: Stack Create Event Done" eventId=dfbb4005-7940-4dc8-87d3-fd56c687c994 resourceId=1st4
```

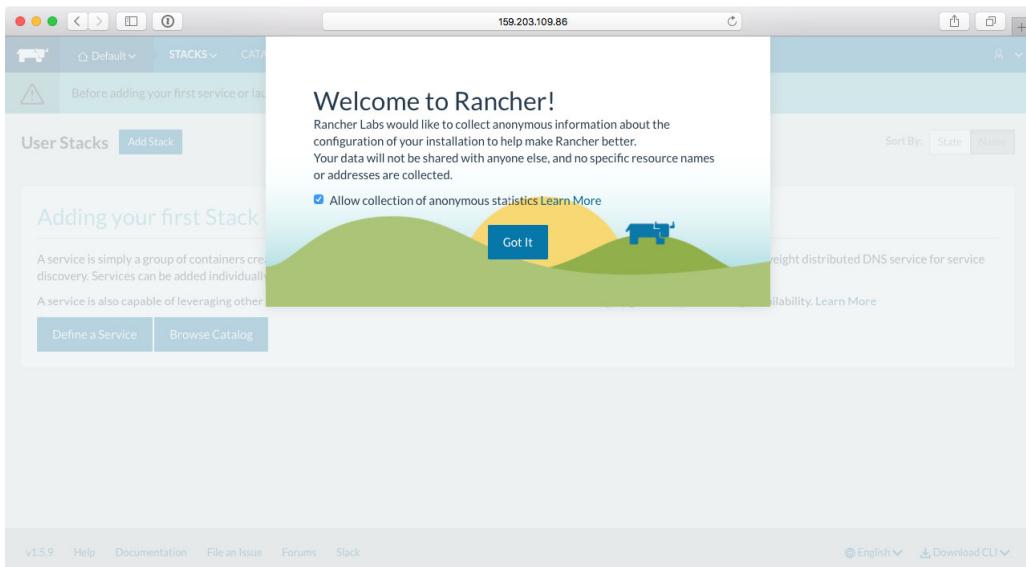
Po zakończeniu procesu instalacji możemy skorzystać z kombinacji klawiszy `Ctrl+C` i zakończyć wyświetlanie wpisów dziennika instalacji w oknie Terminala. Jeżeli korzystasz z systemu macOS, możesz otworzyć interfejs narzędzia Rancher za pomocą następującego polecenia:

```
$ open http://$(docker-machine ip rancher):8080
```

W celu ustalenia adresu IP swojego hosta narzędzia Rancher możesz również skorzystać z polecenia:

```
$ docker-machine ip rancher
```

Po ustaleniu adresu IP wprowadź go w przeglądarce i dodaj port :8080 (uzyskasz w ten sposób adres podobny do następującego: <http://123.123.123.123:8080>). Po załadowaniu strony w przeglądarce zobaczysz następujący komunikat powitalny:



Kliknij przycisk *Got It* (rozumiem), aby przejść dalej.

Konfiguracja uwierzytelniania

Na początku musimy zabezpieczyć naszą instalację, ponieważ jest ona dostępna publicznie. W tym celu należy skorzystać z opcji *Access* (dostęp) znajdującej się w rozwijanym menu *Admin* (administrator). Domyślnie Rancher pyta o chęć używania usługi uwierzytelniającej GitHub (patrz pierwszy zrzut na następnej stronie).

Na ekranie pojawiają się instrukcje ułatwiające proces konfiguracji. Kliknięcie standardowego odnośnika kierującego do serwisu GitHub przeniesie nas na stronę <https://github.com/settings/developers/>. Po wejściu na nią należy wykonywać instrukcje wyświetlane na ekranie aż do uzupełnienia następującego formularza (patrz drugi zrzut na następnej stronie).

Access Control

Before adding your first service or launching a container, you'll need to add a Linux host with a supported version of Docker. Add a host

GITHUB

GitHub is not configured
Rancher can be configured to restrict access to a set of GitHub users and organization members. This is not currently set up, so anybody that reaches this page (or the API) has full control over the system.

1. Setup a GitHub Application

- For standard GitHub, [click here](#) to go to applications settings in a new window.
For GitHub Enterprise, log in to your account. Click on Settings, then Applications.
- Click "Register new application" and fill out the form:

| | |
|-----------------------------|------------------------------------|
| Application name: | Anything you like, e.g. My Rancher |
| Homepage URL: | http://159.203.109.86:8080/ |
| Application description: | Anything you like, optional |
| Authorization callback URL: | http://159.203.109.86:8080/ |
- Click "Register Application"

Personal settings

Profile

Account

E-mails

Notifications

Billing

SSH and GPG keys

Security

Blocked users

Repositories

Organizations

Saved replies

Authorized OAuth Apps

Authorized Integrations

Installed integrations

Register a new OAuth application

Application name

Rancher

Something users will recognize and trust

Homepage URL

http://159.203.109.86:8080/

The full URL to your application homepage

Application description

My test installation of Rancher

This is displayed to all potential users of your application

Authorization callback URL

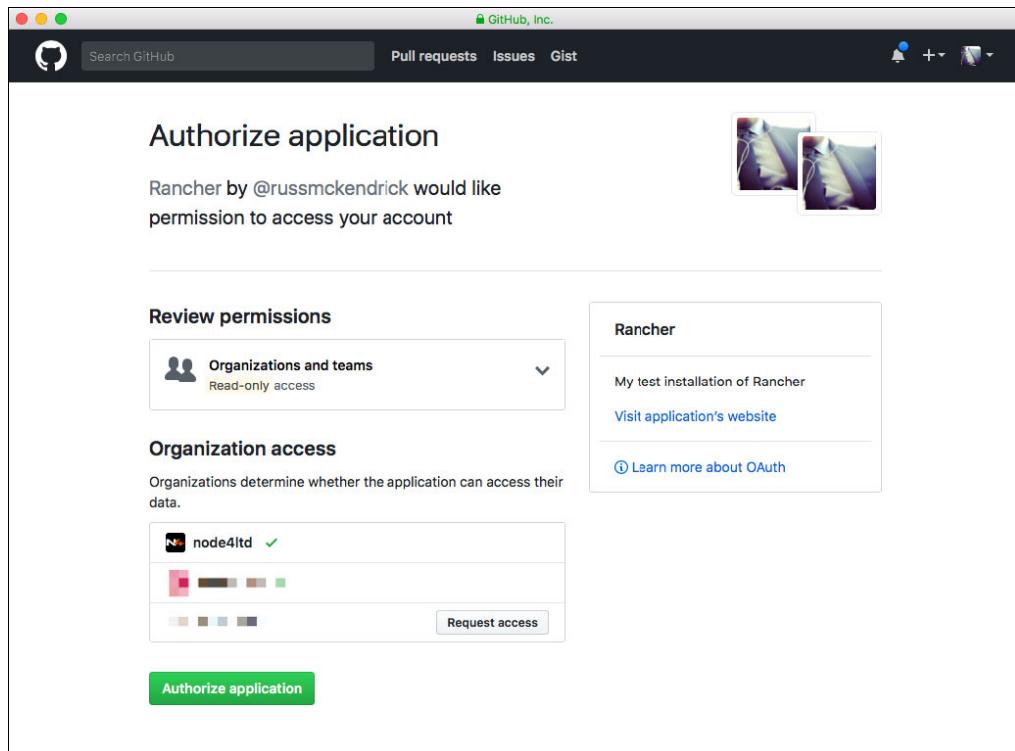
http://159.203.109.86:8080/

Your application's callback URL. Read our [OAuth documentation](#) for more information.

Register application **Cancel**

Adres URL strony domowej i adres URL wywołania zwrotnego autoryzacji należy podać zgodnie z instrukcjami narzędzia Rancher. Podanie tych adresów pozwoli na powrót do usługi Rancher po zalogowaniu się do serwisu GitHub.Więcej informacji na temat aplikacji OAuth i innych aplikacji GitHub znajdziesz na stronie <https://developer.github.com/v3/guides/getting-started/>.

Po zarejestrowaniu aplikacji wpisz dane uwierzytelniające w polach narzędzia Rancher i kliknij przycisk *Save* (zapisz). Po zapisaniu możesz testowo się zalogować. Spowoduje to otwarcie następującego okna:



Po kliknięciu przycisku *Authorize application* (autoryzuj aplikację) strona narzędzia Rancher zostanie odświeżona i pojawi się na niej komunikat *GitHub is enabled* (GitHub został aktywowany) (patrz pierwszy zrzut na następnej stronie).

Po zainstalowaniu podstawowej wersji narzędzia Rancher i zabezpieczeniu go możemy przyjąć się tworzeniu środowiska roboczego i dodawaniu do niego hostów. Z komunikatów wyświetlanych podczas instalacji narzędzia Rancher wynika, że należy to zrobić przed uruchomieniem usług.

The screenshot shows the Rancher web interface with the URL 159.203.109.86 in the address bar. The top navigation bar includes links for Default, STACKS, CATALOG, INFRASTRUCTURE, ADMIN, and API. A warning message at the top states: "Before adding your first service or launching a container, you'll need to add a Linux host with a supported version of Docker. Add a host".

Access Control

GitHub is enabled
Rancher is configured to allow access to environment members, no organizations and 1 user.

Authentication

Client ID: cca6f7753f69ac6add1c
To change the configured GitHub application, disable access control below and then set it up again.

Site Access

Configure who should be allowed to log in and use Rancher.

Allow any valid Users (radio button selected)
Allow members of Environments, plus Authorized Users and Organizations
Restrict access to only Authorized Users and Organizations

Add a GitHub user or organization name: +

Authorized Users and Organizations:
russmckendrick Russ McKendrick

Saved!

Tworzenie stada

Domyślne środowisko utworzone przez narzędzie Rancher korzysta z własnej technologii orkiestracji o nazwie **Cattle** (bydło). Autorzy narzędzia Rancher opisują tę technologię w następujący sposób:

Cattle to silnik orkiestracji narzędzia Rancher. Jego głównym zadaniem jest zarządzanie metadanymi i orkiestracja zewnętrznych systemów. Cattle tak naprawdę nie wykonuje żadnych samodzielnie — przesyła je do innych komponentów, które mogą je wykonać. Cattle można traktować jak pośrednią warstwę zarządzającą narzędzia Rancher. Czy tak naprawdę menedżerowie średniego szczebla wykonują jakieś zadania?

Zanim uruchomimy aplikacje korzystające z technologii Cattle, musimy utworzyć niezbędne hosty. W celu dodania hosta do grupy określonej mianem *herd* (stado) należy wybrać opcję *Hosts* (hosty) z rozwijanego menu *Infrastructure* (infrastruktura). Spowoduje to wyświetlenie prawie pustej strony potwierdzającej brak aktywnych hostów i kontenerów.

Kliknięcie przycisku *Add Host* (dodaj hosta) spowoduje wyświetlenie kilku opcji. Korzystamy z chmury DigitalOcean, a więc kliknij ikonę *DigitalOcean* i wprowadź klucz dostępowy do jej interfejsu:

Hosts: Add Host

Manage available machine drivers

ACCOUNT ACCESS

Access Token*

A Personal Access Token from the DigitalOcean Apps & API screen

Next: Configure Droplet Cancel

Po kliknięciu przycisku *Next: Configure Droplet* (dalej: skonfiguruj aplet chmury) wyświetlna zostanie strona z opcjami apletów tworzących stado (*herd*):

Hosts: Add Host

Manage available machine drivers

Name*

rancher
Hosts will be named rancher1 – rancher3

Quantity

3

REGION

Region New York 3

INSTANCE

Image Ubuntu 16.04.2 x64

Size 0.5 GB RAM, 20 GB Disk, 1 vCPU

SSH User root

Z zaprezentowanego zrzutu wynika, że uruchamiam trzy apletty systemu Ubuntu 16.04 (512 MB) w regionie *New York 3*. Kliknięcie przycisku *Create* (utwórz) spowoduje utworzenie apletów i zarejestrowanie ich w narzędziu Rancher.

Po upływie kilku minut w sekcji *Hosts* powinny znaleźć się trzy uruchomione przez nas hosty, a także opisujące je informacje:

The screenshot shows the Rancher interface with three hosts listed under the 'Hosts' tab:

- rancher1** (Region: New York 3)
 - Stack: healthcheck**: health-check-2 (IP: 10.42.179.73)
 - Stack: ipsec**: ipsec-1 (IP: 10.42.204.244), Sidekicks: ipsec-2, ipsec-3
 - Stack: network-services**: .network-manager-2 (None), .metadata-2 (IP: 172.17.0.2), Sidekicks: .network-manager-3, .metadata-3
 - Standalone Containers**: rancher-agent-bootstrap (None), rancher-agent (None)
- rancher2** (Region: New York 3)
 - Stack: healthcheck**: health-check-3 (IP: 10.42.106.252)
 - Stack: ipsec**: ipsec-2 (IP: 10.42.228.4), Sidekicks: ipsec-3
 - Stack: network-services**: .network-manager-3 (None), .metadata-3 (IP: 172.17.0.2), Sidekicks: .network-manager-4, .metadata-4
 - Stack: scheduler**: scheduler-1 (IP: 10.42.141.108)
 - Standalone Containers**: rancher-agent (None), rancher-agent-bootstrap (None)
- rancher3** (Region: New York 3)
 - Stack: healthcheck**: health-check-4 (IP: 10.42.230.143)
 - Stack: ipsec**: ipsec-3 (IP: 10.42.4.209), Sidekicks: ipsec-2
 - Stack: network-services**: .network-manager-4 (None), .metadata-4 (IP: 172.17.0.2), Sidekicks: .network-manager-5, .metadata-5
 - Standalone Containers**: rancher-agent-bootstrap (None), rancher-agent (None)

Dysponujemy już hostami wchodzącymi w skład grupy (tzw. stada), a więc możemy zająć się uruchamianiem stosu.

Uruchamianie stosów

Rancher obsługuje uruchamianie dwóch różnych typów stosów, takich jak stosy zdefiniowane przez użytkownika i stosy będące infrastrukturą. Stosy to zbiory usług, które składają się z kontenerów. Zwykle usługi te są tak skonfigurowane, aby się ze sobą komunikowały tak jak w przypadku usług definiowanych w plikach Docker Compose.

Stosy infrastruktury rozszerzają możliwości narzędzia Rancher — dodają obsługę zewnętrznych funkcji, takich jak np. mechanizmy równoważenia obciążenia, trwałe magazyny danych i usługi DNS.

Stosy definiowane przez użytkownika

W tej sekcji przyjrzymy się procesowi uruchamiania stosu definiowanego przez użytkownika. Stos taki jest konfigurowany od dołu w górę. Podobnie jak w poprzednich rozdziałach będziemy teraz korzystać z aplikacji cluster, którą można pobrać ze strony <https://hub.docker.com/r/russmckendrick/cluster/>.

Po wybraniu opcji *All* (wszystkie) z menu *STACKS* (stosy) zobaczyś, że w Twoim środowisku działają już jakieś stosy. Obsługują one usługi Rancher i Cattle, z których właśnie korzystasz:

| Stack Name | Status | Services | Containers |
|------------------|------------|------------|--------------|
| healthcheck | Up to date | 1 Services | 3 Containers |
| ipsec | Up to date | 1 Services | 9 Containers |
| network-services | Up to date | 2 Services | 9 Containers |
| scheduler | Up to date | 1 Service | 1 Container |

Po kliknięciu przycisku *Add Stack* (dodaj stos) na ekranie pojawi się prosty formularz definiowania stosu. W celu utworzenia stosu należy podać jego nazwę i opis. Ponadto formularz zawiera opcje importowania plików Docker Compose i Rancher Compose, ale na razie nie będziemy z nich korzystać. W moim przykładzie wprowadziłem następujące dane:

- *Name* (nazwa): MyClusterApplication (swojej aplikacji możesz nadać dowolną nazwę, ale nie może ona zawierać spacji ani innych znaków specjalnych);
- *Description* (opis): Testing Rancher with the cluster application (testowanie narzędzia Rancher za pomocą aplikacji cluster).

Oto formularz, w którym wprowadziłem powyższe dane:

Add Stack

| | |
|------------------------|--|
| Name | Description |
| My Cluster Application | Testing Rancher with the cluster application |

OPTIONAL: IMPORT COMPOSE

| | |
|--------------------------------|---------------------------------|
| Optional: docker-compose.yml | Optional: rancher-compose.yml |
| Contents of docker-compose.yml | Contents of rancher-compose.yml |

ADVANCED OPTIONS ▾

Create **Cancel**

Po wprowadzeniu danych do formularza kliknij przycisk *Create* (utwórz). Na ekranie pojawią się dane opisujące nowy stos (na razie jest ich niewiele):

The screenshot shows the Rancher web interface. At the top, there's a navigation bar with links for Default, STACKS, CATALOG, INFRASTRUCTURE, ADMIN, and API. Below the navigation is a toolbar with icons for creating services, stacks, and infrastructure. A dropdown menu labeled 'Stack' is open, showing 'MyClusterApplication'. To the right of the stack name is a button labeled 'Add Service'. Below the stack name, there's a 'Description' field containing the text 'Testing Rancher with the cluster application'. Underneath the description, it says 'No Services'.

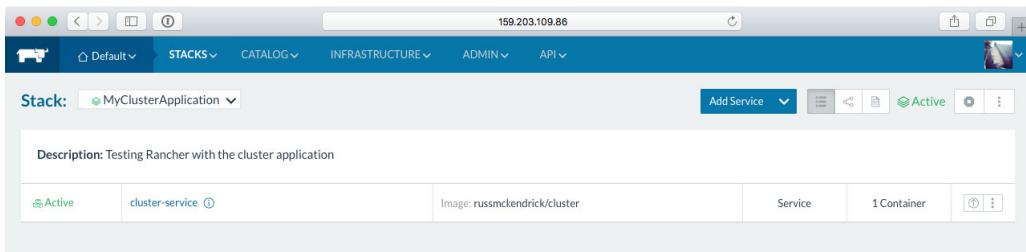
Kolejnym etapem uruchamiania stosu jest dodanie kontenera klastra. W tym celu należy kliknąć przycisk *Add Service* (dodaj usługę). Wyświetlony formularz skonfigurowalem w następujący sposób:

- *Scale (skala): Run 1 container* (uruchom 1 kontener);
- *Name (nazwa): cluster-service* (usługa klastra);
- *Description (opis): The containers running russmckendrick/cluster* (kontenery aplikacji russmckendrick-cluster);
- *Select Image (wybierz obraz): russmckendrick/cluster*;
- *Always pull images before creating* (zawsze pobieraj obrazy przed utworzeniem kontenera): tę opcję należy pozostawić zaznaczoną.

Oto wygląd uzupełnionego przeze mnie formularza:

The screenshot shows the 'Add Service' dialog. At the top, it has a 'Scale' section with a radio button selected for 'Run 1 container'. Below that is a checkbox for 'Always run one instance of this container on every host'. The service name is set to 'cluster-service' and the description is 'The containers running russmckendrick/cluster'. Under the 'Select Image*' section, the image 'russmckendrick/cluster' is chosen, and a checkbox for 'Always pull image before creating' is checked. There are also sections for 'Port Map' and 'Service Links' at the bottom.

Nie zmieniaj pozostałych opcji i kliknij przycisk *Create* (utwórz) znajdujący się na dole strony. Spowoduje to uruchomienie kontenera.

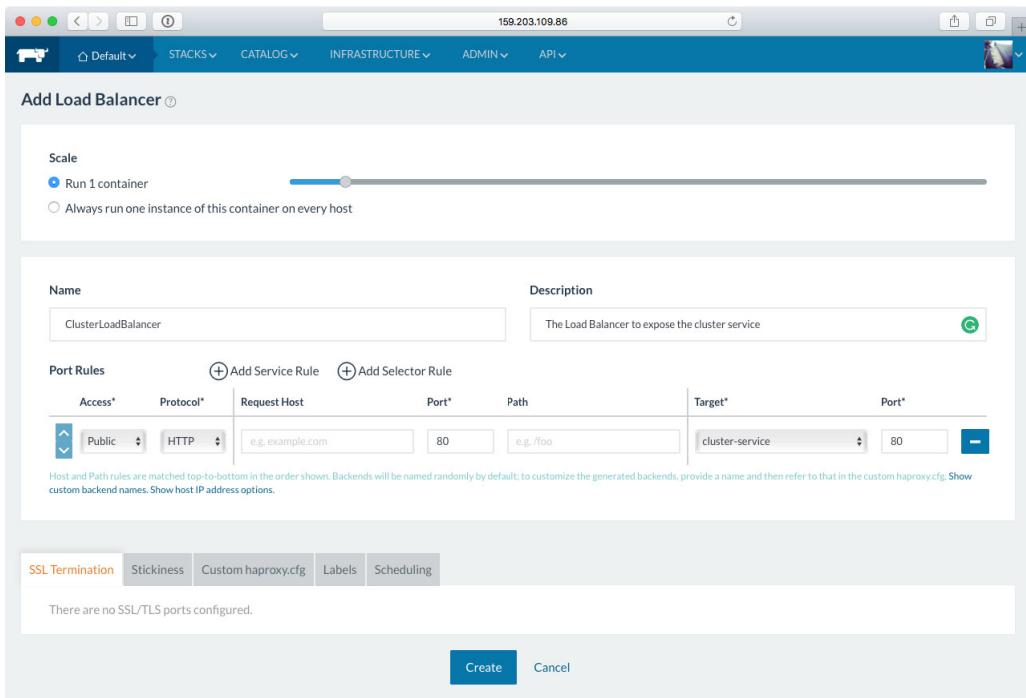


The screenshot shows the Rancher interface with the URL 159.203.109.86. In the top navigation bar, there are tabs for Default, STACKS, CATALOG, INFRASTRUCTURE, ADMIN, and API. Below the navigation, a stack named 'MyClusterApplication' is selected. A description below the stack says 'Testing Rancher with the cluster application'. Under the stack, there is a table with one row. The first column is 'Active', the second is 'cluster-service' with a help icon, the third is 'Image: russmcclendrick/cluster', the fourth is 'Service', and the fifth is '1 Container'.

Pomimo uruchomienia usługi wciąż nie będziemy mogli uzyskać do niej dostępu z zewnątrz, ponieważ musimy uruchomić jeszcze mechanizm równoważenia obciążenia. Kliknij strzałkę w dół widoczną na przycisku *Add Service* (dodaj usługę) i wybierz opcję *Add Load Balancer* (dodaj mechanizm równoważenia obciążenia). Oto dane, jakie wprowadziłem podczas konfiguracji mojego mechanizmu:

- *Name* (nazwa): cluster-loadbalancer (równoważenie obciążenia klastra);
- *Description* (opis): The Load Balancer to expose the cluster service (mechanizm równoważenia obciążenia udostępniający klaster);
- *Port Rules* (reguły portów): dodaj port nr 80, a następnie wybierz usługę cluster-service.

Pozostaw domyślne ustawienia pozostałych opcji i kliknij przycisk *Create*:

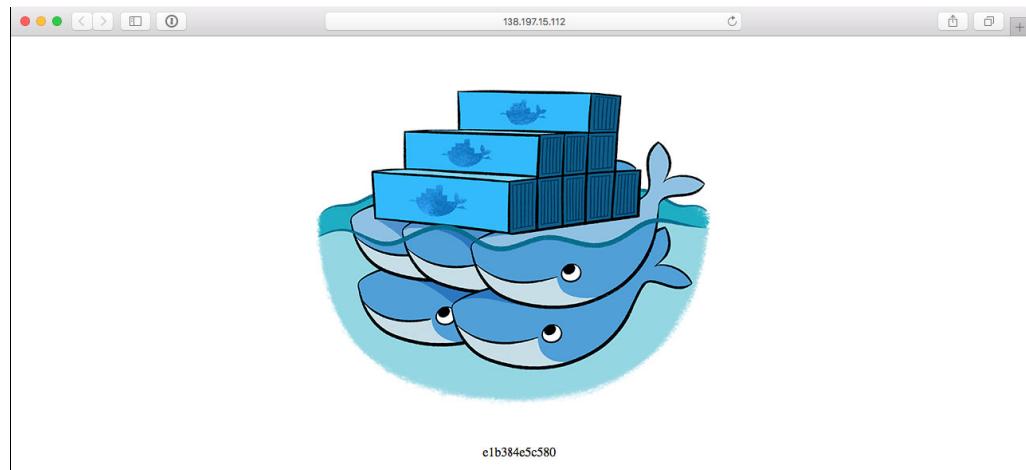


The screenshot shows the 'Add Load Balancer' dialog. At the top, under 'Scale', the radio button 'Run 1 container' is selected. Below that, there's a 'Name' field containing 'ClusterLoadBalancer' and a 'Description' field with the text 'The Load Balancer to expose the cluster service'. Under 'Port Rules', there's a table with columns: Access*, Protocol*, Request Host, Port*, Path, Target*, and Port*. One row is shown with 'Access*' set to 'Public', 'Protocol*' to 'HTTP', 'Request Host' to 'e.g. example.com', 'Port*' to '80', 'Path' to 'e.g. /foo', 'Target*' to 'cluster-service', and 'Port*' to '80'. A note at the bottom of this section says 'Host and Path rules are matched top-to-bottom in the order shown. Backends will be named randomly by default; to customize the generated backends, provide a name and then refer to that in the custom haproxy.cfg. Show custom backend names. Show host IP address options.' At the bottom of the dialog, there are tabs for 'SSL Termination', 'Stickiness', 'Custom haproxy.cfg', 'Labels', and 'Scheduling'. The 'SSL Termination' tab is active. Below these tabs, a message says 'There are no SSL/TLS ports configured.' At the very bottom are 'Create' and 'Cancel' buttons.

Teraz nasz stos składa się z dwóch aktywnych usług: *cluster-loadbalancer* i *cluster-service* — powinniśmy mieć dwa kontenery:

| Stack: | MyClusterApplication | Add Service | Active | |
|----------|--------------------------------------|-----------------------------------|---------------|-------------|
| - Active | cluster-loadbalancer | To: cluster-service Ports: 80/tcp | Load Balancer | 1 Container |
| Active | cluster-service | Image: russmckendrick/cluster | Service | 1 Container |

Kliknięcie odnośnika *80/tcp* usługi *cluster-loadbalancer* spowoduje otwarcie nowej zakładki przeglądarki, w której wyświetlona zostanie nasza aplikacja o nazwie *cluster*:



Wróć do panelu narzędzia Rancher i kliknij ikonę informacji (*Info*). Spowoduje to wysunięcie z dołu lub z boku ekranu pola z informacjami na temat klastra *cluster-loadbalancer*. Teraz możesz przeprowadzić skalowanie usługi. Usługę skonfigurowałem tak, aby działała ona w sześciu kontenerach (patrz pierwszy zrzut na następnej stronie).

Odświeżając stronę aplikacji, możesz obserwować zmiany identyfikatora kontenera (przejdz przez wszystkie sześć kontenerów). Wróć do panelu narzędzia Rancher. Zauważ trzy przyciski znajdujące się pomiędzy przyciskiem *Add Service* i statusem stosu (stos powinien być teraz aktywny — *Active*). Służą one do zmiany widoku bieżcej strony (pierwszy przycisk aktywuje obecny widok listy).

Stack: MyClusterApplication

Description: Testing Rancher with the cluster application

| Active | cluster-loadbalancer | To: cluster-service Ports: 80/tcp | Load Balancer | 1 Container |
|--------|----------------------|-----------------------------------|---------------|--------------|
| Active | cluster-service | Image: russmckendrick/cluster | Service | 6 Containers |

Info (View Details)

Active
Image: russmckendrick/cluster
Entrypoint: None
Command: None
Description: The containers running russmckendrick/cluster

Containers (6)
Scale 6

Ports
No Public Ports

Links
No Links

Drugi ze wspomnianych przycisków włącza widok grafu. W widoku tym przedstawiana jest topologia stosu i sposób połączenia ze sobą usług. Nasz stos składa się zaledwie dwóch usług, a więc zostanie on przedstawiony za pomocą dwóch pól połączonych linią. Ostatni z przycisków aktywuje widok składowych plików YAML. Wybranie go spowoduje wyświetlenie zawartości plików *docker-compose.yml* i *rancher-compose.yml* użytych podczas tworzenia stosu w jego obecnej konfiguracji:

```
version: '2'
services:
  cluster-service:
    image: russmckendrick/cluster
    stdin_open: true
    tty: true
    labels:
      io.rancher.container.pull_image: always
  cluster-loadbalancer:
    image: rancher/ce-service-loadbalancer:v0.6.9
    ports:
    - :80/tcp
    labels:
      io.rancher.container.agent.role: environmentAdmin
      io.rancher.container.create_agent: 'true'
```

```
version: '2'
services:
  cluster-service:
    scale: 6
    start_on_create: true
  cluster-loadbalancer:
    scale: 1
    start_on_create: true
    lb_config:
      certs: []
      port_rules:
      - path: ''
        priority: 1
        protocol: Http
        service: cluster-service
        source_port: 80
        target_port: 80
    health_check:
      response_timeout: 2000
      healthy_threshold: 2
      port: 42
      unhealthy_threshold: 3
      initializing_timeout: 60000
      interval: 2000
      reinitializing_timeout: 60000
```

Plik *docker-compose.yml* jest plikiem składowym w standardzie V2, który definiuje nasze dwie usługi:

```
version: '2'
services:
  cluster-service:
    image: russmckendrick/cluster
    stdin_open: true
    tty: true
    labels:
      io.rancher.container.pull_image: always
  cluster-loadbalancer:
    image: rancher/lb-service-haproxy:v0.6.4
    ports:
      - 80:80/tcp
    labels:
      io.rancher.container.agent.role: environmentAdmin
      io.rancher.container.create_agent: 'true'
```

Jak widzisz, wszystkie opcje narzędzia Rancher są zdefiniowane przy użyciu etykiet. Plik *rancher-compose.yml* zawiera wszystkie metadane użyte w celu zdefiniowania bieżącego stanu stosu:

```
version: '2'
services:
  cluster-service:
    scale: 6
    start_on_create: true
  cluster-loadbalancer:
    scale: 1
    start_on_create: true
    lb_config:
      certs: []
      port_rules:
        - path: ''
          priority: 1
          protocol: http
          service: cluster-service
          source_port: 80
          target_port: 80
      health_check:
        response_timeout: 2000
        healthy_threshold: 2
        port: 42
        unhealthy_threshold: 3
        initializing_timeout: 60000
        interval: 2000
        reinitializing_timeout: 60000
```

Klikając znajdujący się na końcu po prawej stronie przycisk oznaczony symbolem wielokropka i wybierając z rozwijanego menu opcję *Export Config* (eksportuj konfigurację), możesz pobrać pliki definiujące stos.

Podgląd właściwości kontenerów

Wybranie opcji *Containers* (kontenery) z menu *INFRASTRUCTURE* (infrastruktura) spowoduje wyświetlenie listy wszystkich kontenerów uruchomionych w grupie (stadzie). Po wprowadzeniu hasła *cluster* w polu wyszukiwania lista zostanie przefiltrowana i zostaną na niej po-zostawione tylko kontenery wchodzące w skład naszego stosu:

| Actions | State | Name | IP Address | Host | Image | Command |
|--------------------------|---------|---------------------------------|---------------|----------|---------------------------------|---------|
| <input type="checkbox"/> | Running | MyClusterApplication-cluster... | 10.42.49.206 | rancher1 | rancher/lb-service-haproxy:v... | None |
| <input type="checkbox"/> | Running | MyClusterApplication-cluster... | 10.42.200.135 | rancher1 | russmkendrick/cluster | None |
| <input type="checkbox"/> | Running | MyClusterApplication-cluster... | 10.42.244.112 | rancher3 | russmkendrick/cluster | None |
| <input type="checkbox"/> | Running | MyClusterApplication-cluster... | 10.42.75.60 | rancher1 | russmkendrick/cluster | None |
| <input type="checkbox"/> | Running | MyClusterApplication-cluster... | 10.42.56.20 | rancher1 | russmkendrick/cluster | None |

Kliknięcie nazwy kontenera (np. *MyClusterApplication-cluster-loadbalancer-1*) spowoduje wyświetlenie szczegółowych informacji na temat kontenera. Wśród nich znajdziesz wykresy obciążenia procesora (*CPU*), a także wykorzystania zasobów pamięci (*Memory*), sieci (*Network*) i przestrzeni dyskowej (*Storage*):

Container: MyClusterApplication-cluster-loadbalancer-1

Stack/Service: MyClusterApplication/cluster-loadbalancer

Host: rancher1

Container IP: 10.42.49.206

Docker ID: e0cdb4fde889...

Image: rancher/lb-service-haproxy:v0.6.4

CPU

Memory

Network

Storage

Ports **Command** **Volumes** **Networking** **Security** **Health Check** **Labels** **Scheduling**

| IP Address | Public on Host | Private in Container | Protocol |
|------------|----------------|----------------------|----------|
| 0.0.0.0 | 80 | 80 | TCP |

Przycisk oznaczony symbolem wielokropka udostępnia opcję ponownego uruchomienia (*Restart*), zatrzymania (*Stop*), skasowania (*Delete*) i wykonania powłoki (*Execute Shell*).

Katalog

Rancher umożliwia uruchamianie uprzednio zdefiniowanych stosów. W narzędziu to wbudowano kilkanaście różnych predefiniowanych aplikacji. Znajdziesz je w menu *Catalog* (katalog). Oto zrzut prezentujący niektóre z aplikacji dostępnych w katalogu:

The screenshot shows the Rancher web interface with the 'CATALOG' tab selected in the top navigation bar. Below it, a grid displays eight application stacks:

- Alfresco**: An ECM and BPM platform. (View Details)
- Apache Kafka**: (Experimental) Kafka cluster. (View Details)
- Apache Zookeeper**: (Experimental) Zookeeper cluster. (View Details)
- asclinema-org**: Asciinema is a free and open source solution for recording terminal sessions and sharing them on the web. (View Details)
- Bind9 Domain Name Server**: Bind9 DNS server, compatible with the "DNS Update (RFC2136)". (View Details)
- Cloud9**: (Experimental) Cloud 9 SDK. (View Details)
- CloudFlare DNS**: Rancher External DNS service powered by CloudFlare. (View Details)
- Concrete5**: Concrete5.7 CMS for building easy and beautiful websites. (View Details)

Rozwiązywanie to pozwala na szybkie uruchamianie i konfigurowanie stosów obsługujących wybrane aplikacje. Podczas uruchamiania niektórych aplikacji z tego katalogu warto pamiętać o możliwości wystąpienia problemów spowodowanych słabą specyfikacją hostów naszego stada. Utworzyliśmy proste środowisko testowe, w którym nie można uruchomić aplikacji opartych na kodzie Java, takich jak np. Jenkins.

Usuwanie stada

Zanim przejdziemy do kolejnego podrozdziału, powinniśmy usunąć hosty z naszego stada. Zamiast wykonywania tej operacji za pomocą narzędzia Rancher zalecam otwarcie panelu sterowania chmury DigitalOcean i użycie go w celu bezpośredniego zakończenia pracy apletów.

W przeszłości musiałem ponosić dodatkowe koszty z powodu błędów upłynięcia czasu żądania, które przyczyniały się do niepełnego zamknięcia hostów.

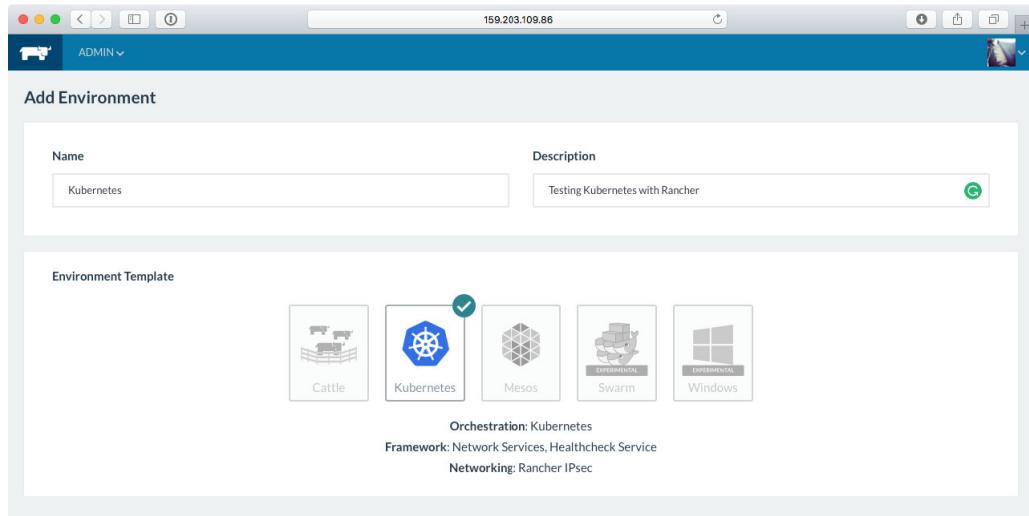
Zakończ wszystkie trzy hosty — Rancher1, Rancher2 i Rancher3, a następnie sprawdź, czy nie pojawiają się one na liście panelu sterowania chmury DigitalOcean.

Inne klastry

Rancher umożliwia również uruchamianie innych klastrów. Obsługuje klastry:

- Kubernetes,
- Mesos,
- Docker Swarm (na razie jest to funkcja eksperymentalna).

Rancher umożliwia bezpośrednie uruchamianie tych klastrów, a także przełączanie się pomiędzy różnymi środowiskami klastrów. Tworzenie nowego środowiska jest bardzo proste. Wystarczy kliknąć przycisk *Default* (ustawienia domyślne), wybrać opcję *Environment* (środowisko), a następnie kliknąć *Add Environment* (dodaj środowisko). Na ekranie pojawi się wtedy prosta strona, na której należy wprowadzić nazwę środowiska i wybrać jego szablon:



Po utworzeniu środowiska można do niego dodawać hosty — wystarczy skorzystać z zaprezentowanej wcześniej techniki:

Hosts: Add Host

Name* Quantity

Manage available machine drivers

Hosts will be named Kubernetes1 – Kubernetes5

Dodanie nowych hostów może trwać nawet 10 minut (czas ten zależy od wybranego szablonu środowiska), ponieważ narzędzie Rancher musi zainstalować i skonfigurować klienta i sieć, a także uruchomić środowisko klastra.

Po zakończeniu wykonywania operacji Rancher usunie większość opcji dostępnych w górnym menu. Wynika to z tego, że Rancher nie jest graficznym interfejsem przeznaczonym do zarządzania środowiskiem. W związku z tym podczas uruchamiania np. klastra Kubernetes instalowany jest *Kubernetes Dashboard* (panel sterowania Kubernetes):

Kubernetes Dashboard

 Dashboard

Kubernetes Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster and troubleshoot them, as well as manage the cluster itself.

[Kubernetes UI](#)

Kliknięcie przycisku *Kubernetes UI* (interfejs użytkownika środowiska Kubernetes) spowoduje otwarcie panelu sterowania (logowanie do tego panelu przebiega automatycznie, gdy użytkownik jest zalogowany do narzędzia Rancher) (patrz pierwszy zrzut na następnej stronie).

Istnieją również inne narzędzia przeznaczone do zarządzania klasteryami Kubernetes, takie jak np. interfejs tekstowy *kubectl*. Narzędzie Rancher umożliwia uzyskanie dostępu do kontenera tego interfejsu z poziomu Terminala, a także pozwala na pobranie jego konfiguracji, która może być użyta podczas instalowania narzędzia *kubectl* w środowisku lokalnym (patrz drugi zrzut na następnej stronie).

Docker. Programowanie aplikacji dla zaawansowanych

The screenshot shows the Kubernetes Admin interface at the URL 159.203.87.167. On the left, a sidebar lists various administrative sections: Namespaces, Nodes (selected), Persistent Volumes, Namespace (default), Workloads, Deployments, Replica Sets, Replication Controllers, Daemon Sets, Stateful Sets, Jobs, Pods, Services and discovery, Services, Ingresses, Storage, Persistent Volume Claims, and Config. The main area displays two line charts: 'CPU usage' and 'Memory usage'. The 'CPU usage' chart shows a sharp peak around 14:35. The 'Memory usage' chart shows a steady trend from 1.82 GiB to 2.42 GiB. Below the charts is a table titled 'Nodes' listing two nodes: 'kubernetes1' and 'kubernetes2'. Each node entry includes its name, labels (beta.kubernetes.io/arch: amd64, beta.kubernetes.io/instance-type: ran..., beta.kubernetes.io/os: linux, failure-domain.beta.kubernetes.io/region:..., failure-domain.beta.kubernetes.io/zone:...), and status (Ready: True, Age: 38 minutes). There is also a 'show all labels' link.

The screenshot shows the Kubernetes CLI interface at the URL 159.203.87.167. At the top, there are dropdown menus for KUBERNETES, INFRASTRUCTURE, ADMIN, and API. A 'Generate Config' button is visible. Below it, a message says: 'To use kubectl (v1.4+ only) on your workstation, click the button to generate an API key and config file:'. A terminal window titled 'Shell: kubernetes-kubectl-1' shows the output of the 'top' command for Kubernetes components. The output includes columns for NAME, CPU Requests, CPU Limits, Memory Requests, and Memory Limits. It shows pods like 'heapster-818085469-q615g', 'kubernetes-dashboard-2492700511-j4qcg', and 'tiller-deploy-3991468440-j285d'. The terminal also displays events such as 'Starting kubelet' and 'ImageGCFailed'. At the bottom, a message says 'Connected'.

Pozostałe szablony środowisk działają w ten sam sposób. Szablon Docker Swarm wdraża narzędzie Portainer, które umożliwia zarządzanie klastrem Swarm.

Podsumowanie

Na początku rozdziału stwierdziłem, że Rancher na pozór przypomina swym wyglądem narzędzie Portainer. Mam nadzieję, że po przeczytaniu tego rozdziału wiesz, że są to zupełnie inne narzędzia.

Rancher oferuje te same funkcje, gdy korzysta się z wbudowanej w niego technologii orkiestracji (Cattle). Jej zaletą jest prostota uruchamiania rozbudowanych klastrów w wielu chmurach i możliwość korzystania z wszystkich opcji oferowanych przez wybrane środowisko. Ręczne uruchamianie wszystkich środowisk obsługiwanych przez narzędzie Rancher nie byłoby łatwym zadaniem.

Nie opisałem interfejsu API narzędzia Rancher, a także wielu innych związanych z nim zagadnień. Każda operacja wykonana za pomocą graficznego interfejsu narzędzia Rancher może być wykonana również za pośrednictwem interfejsu programistycznego, a więc narzędzie Rancher może być integrowane z własnymi mechanizmami przepływu zadań. Więcej informacji na temat interfejsu programistycznego narzędzia Rancher znajdziesz w dokumentacji znajdującej się na stronie <http://rancher.com/docs/rancher/v1.6/en/api/v2-beta/>.

W kolejnym rozdziale zajmiemy się usługą Docker Cloud.

Usługa Docker Cloud

Chmura Docker Cloud jest usługą dostarczaną przez autorów Dockera. Umożliwia ona uruchamianie hostów Dockera i zarządzanie nimi w środowisku publicznym. U podstaw chmury Docker Cloud znajduje się narzędzie o nazwie **Tutum**, które zostało wykupione przez autorów Dockera w październiku 2015 r.

Więcej informacji na temat tego przejęcia znajdziesz w artykule <https://blog.docker.com/2015/10/docker-acquires-tutum/>.

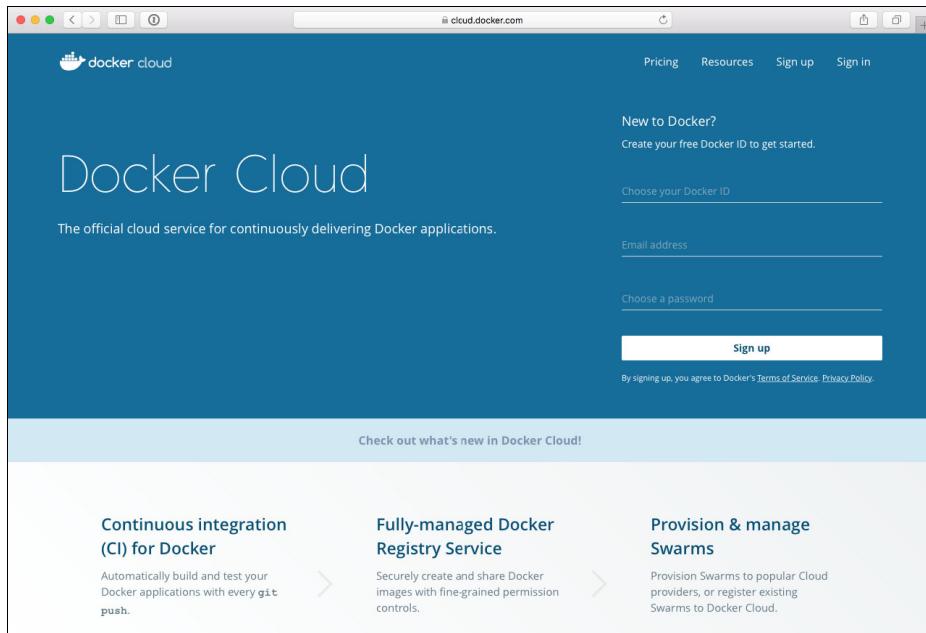
Początkowo narzędzie Tutum miało pomagać w uruchamianiu kontenerów w chmurze — miało być zunifikowanym interfejsem obsługującym różnych dostawców usług chmury. Twórcy Dockera chcieli rozwijać ten pomysł dalej, ale zdecydowali się na przemianowanie go na Docker Cloud. W tym rozdziale zajmiemy się usługą Docker Cloud, a także Dockerem w chmurze Amazon Web Services. Poruszmy następujące zagadnienia:

- zakładanie konta,
- łączenie usług różnych dostawców,
- uruchamianie węzłów,
- tryb Swarm,
- Docker w chmurze Amazon Web Services.

Docker Cloud jest płatną usługą. Obecnie miesięczny koszt utrzymania jednego węzła to 15 dolarów (równowartość około 50 zł). Podczas opisywania kolejnych operacji będę zaznaczał, co wiąże się z ponoszeniem kolejnych kosztów.

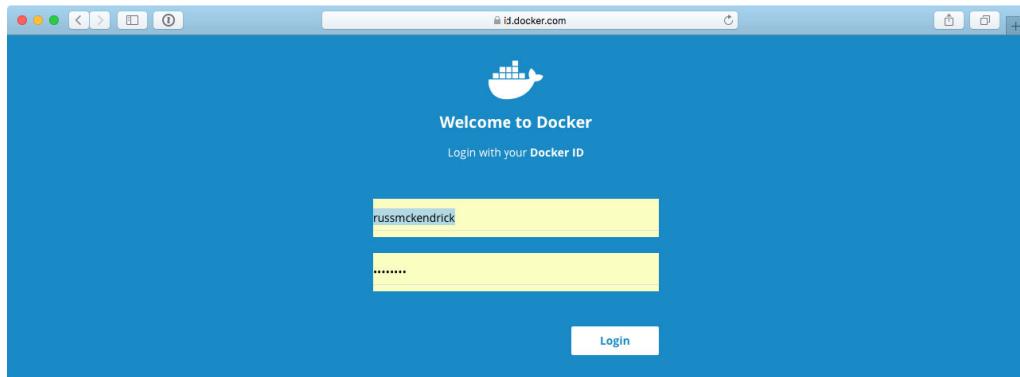
Zakładanie konta

Po wejściu na stronę <https://cloud.docker.com> zobaczysz od razu formularz zakładania nowego konta:



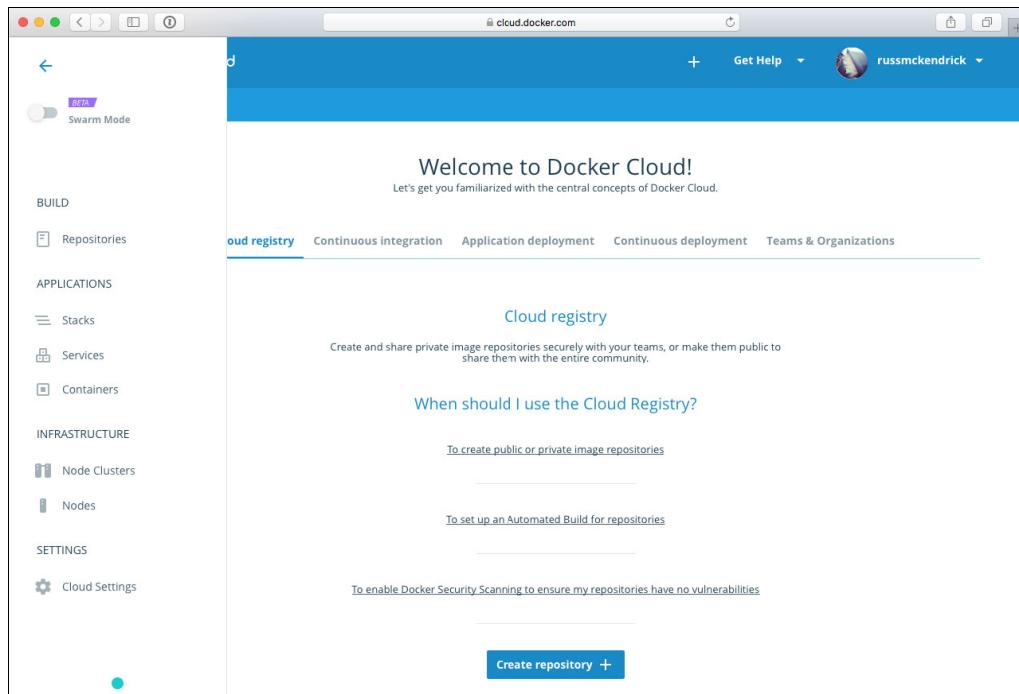
W formularzu tym należy wybrać identyfikator Docker ID. Jeżeli wykonywałeś operacje opisywane w poprzednich rozdziałach tej książki, to prawdopodobnie już masz ten identyfikator. Twórcy Dockera korzystają z tego samego identyfikatora we wszystkich swoich produktach. W związku z tym, jeżeli korzystałeś wcześniej z serwisu Docker Hub, to możesz teraz zalogować się do usługi Docker Cloud.

Kliknij przycisk *Sign in* (zaloguj), a zobaczysz znajomy formularz logowania:



Jeżeli nie masz jeszcze identyfikatora Docker ID, to musisz go najpierw założyć za pomocą formularza dostępnego na głównej stronie usługi Docker Cloud.

Po zalogowaniu się za pomocą identyfikatora Docker ID zobaczysz pusty ekran, a po jego lewej stronie będzie znajdować się wiele opcji. Na poniższym zrzucie rozwinąłem dostępne opcje w celu pokazania ich zawartości:



Zanim skorzystamy z opcji *BUILD* (budowa obrazów), *APPLICATIONS* (aplikacje) lub *INFRASTRUCTURE* (infrastruktura), musimy wejść w menu *Cloud Settings* (ustawienia chmury) w celu połączenia chmury Docker Cloud z naszymi kontami publicznymi.

Łączenie kont

Po pierwszym otwarciu strona *Cloud Settings* będzie wyglądała tak (patrz pierwszy zrzut na następnej stronie).

Znajduje się na niej lista kilku dostawców usług typu chmura. W dalszej części tego podrzędu opiszemy łączenie kont chmur DigitalOcean i Amazon Web Services. Zaczniemy od prostszej operacji — podłączenia chmury DigitalOcean.

The screenshot shows the Docker Cloud web interface. On the left is a sidebar with icons for Swarm Mode, Beta, General, Cloud providers, Source providers, Notifications, Default Privacy, Billing, Plan, and Quotas. The main area has a blue header 'docker cloud'. Below it, 'Cloud Settings' is selected. A profile picture of Russ Kendrick is shown with the text 'Russ Kendrick Member since Feb 15, 2014'. A section titled 'Cloud providers' lists several options with their status and tier: Amazon Web Services (Free Tier), Digital Ocean (\$20 Code), Microsoft Azure (Free trial), SoftLayer (Free trial), and Packet (\$25 code). Each row has an 'Add new credentials' button.

DigitalOcean

W celu powiązania usługi Docker Cloud z kątem wybranej chmury należy kliknąć ikonę przedstawiającą przekreśloną wtyczkę zasilającą. Kliknięcie takiej ikony znajdującej się obok opcji *DigitalOcean* spowoduje otwarcie strony autoryzacji dostępu do aplikacji DigitalOcean.

Jeżeli nie jesteś zalogowany do panelu sterowania chmury DigitalOcean (znajduje się on pod adresem <https://cloud.digitalocean.com/login>), to zostaniesz poproszony o zalogowanie się. Po zalogowaniu się zobaczysz ekran, na którym będziesz mógł udzielić pozwolenia na przeprowadzanie operacji zapisu i odczytu danych przez usługę Docker Cloud:

The screenshot shows a browser window for DigitalOcean, LLC. The title bar says 'DigitalOcean, LLC'. The main content is titled 'Authorize Docker Cloud' with a sub-section 'Read & write Access'. It features an icon of a document with a pencil. Below it, the text 'Docker Cloud would like permission to access your account.' is followed by a checked checkbox next to the email 'russmckendrick@me.com'. At the bottom are two buttons: 'Cancel' and 'Authorize application'.

Zaznacz pole znajdujące się obok swojej nazwy użytkownika, a następnie kliknij przycisk *Authorize application* (autoryzuj aplikację). Po wykonaniu tej operacji na ekranie ponownie pojawi się ekran konfiguracji usługi Docker Cloud.

Amazon Web Services

Łączenie konta Amazon Web Services z usługą Docker Cloud jest nieco bardziej skomplikowane. W celu wykonania tej operacji musimy usłudze Docker Cloud przypisać rolę *Amazon Resource Names* (nazwy zasobów Amazon). Rola ta posiada wszystkie przywileje niezbędne do uzyskania dostępu do zasobów konta Amazon Web Services, które pozwalają na uruchamianie instancji aplikacji.

Musimy zacząć od utworzenia zasady. W tym celu zalogujmy się do konta Amazon Web Services za pośrednictwem strony <https://console.aws.amazon.com/> lub spersonalizowanej strony logowania używanej w danej firmie. Po zalogowaniu się wejdź do menu *Services* (usługi), które znajduje się w lewym górnym rogu strony, i znajdź usługę *IAM*.

Usługa AWS IAM — Identity and Access Management — umożliwia zarządzanie kontami użytkowników i uprawnieniami dostępu do interfejsu programistycznego API chmury AWS na bardzo szczegółowym poziomie. Usługa ta pozwala na tworzenie profili umożliwiających uzyskanie szerokiego dostępu do Twojego konta, a więc nie upubliczniaj danych żadnych profili utworzonych w tej ułudzie. Więcej informacji na ten temat znajdziesz na stronie <https://aws.amazon.com/iam/>.

Po wejściu na stronę *Welcome to Identity and Access Management* (witaj w usłudze zarządzania kontami użytkowników i uprawnieniami dostępu) kliknij przycisk *Polices* (reguły), który znajduje się w menu wyświetlnym po lewej stronie. Na stronie tej znajdziesz wszystkie domyślne reguły zabezpieczeń dostarczone przez Amazon, a także reguły, które zostały utworzone przez Ciebie.

Utwórz własną regułę. Zaczni od kliknięcia przycisku *Create Policy* (utwórz regułę). Spowoduje to otwarcie strony z trzema opcjami: *Copy an AWS Managed Policy* (skopiuj regułę zarządzania AWS), *Policy Generator (generator reguł)* i *Create Your Own Policy* (utwórz własną regułę). Kliknij przycisk *Select* (wybierz) znajdujący się obok opcji *Create Your Own Policy*.

W polu *Policy Name* (nazwa reguły) podaj nazwę reguły. W przykładzie użyłem przykładowej nazwy *dockercloud-policy*. Nie wpisuj niczego w polu *Description* (opis), a w polu *Policy Document* (dokument reguły) wprowadź następujący kod:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:*",
                "iam>ListInstanceProfiles"
            ],
            "Effect": "Allow",
        }
    ]
}
```

```
        "Resource": "*"
    }
]
}
```

Na poniższym zrzucie zaprezentowano wypełniony przeze mnie formularz:

The screenshot shows the 'Review Policy' step in the AWS IAM console. On the left, a sidebar lists steps: 'Step 1 : Create Policy', 'Step 2 : Set Permissions', and 'Step 3 : Review Policy'. The main area is titled 'Review Policy' and contains the following information:

- Policy Name:** dockercloud-policy
- Description:** (Empty text area)
- Policy Document:** A code editor showing a JSON-based policy document:

```
1- {
2-     "Version": "2012-10-17",
3-     "Statement": [
4-         {
5-             "Action": [
6-                 "ec2:*",
7-                 "iam>ListInstanceProfiles"
8-             ],
9-             "Effect": "Allow",
10            "Resource": "*"
11        }
12    ]
13}
```
- Buttons at the bottom:** 'Use autoformatting for policy editing' (checkbox), 'Cancel', 'Validate Policy', 'Previous', 'Create Policy'.

Po wypełnieniu formularza kliknij przycisk *Create Policy* (utwórz regułę). Teraz musisz podać tę regułę do roli. W tym celu kliknij przycisk *Roles* (role) znajdujący się w menu wyświetlanym po lewej stronie, a następnie kliknij przycisk *Create new role* (utwórz nową rolę).

Proces tworzenia roli składa się z czterech etapów. W pierwszym należy wybrać typ tworzonej roli. W celu umożliwienia usłudze Docker Cloud uzyskania dostępu do konta Amazon Web Services wybierz opcję *Role for cross-account access* (rola umożliwiająca dostęp z innego konta):

The screenshot shows the 'Create role' step in the AWS IAM console. On the left, a sidebar lists steps: 'Step 1 : Select role type', 'Step 2 : Establish trust', 'Step 3 : Attach policy', and 'Step 4 : Set role name and review'. The main area is titled 'Select role type' and contains the following options:

- AWS Service Role
- AWS service-linked role
- Role for cross-account access
 - Provide access between AWS accounts you own
 - Allows IAM users from one of your other AWS accounts to access this account.
 - Select button
 - Provide access between your AWS account and a 3rd party AWS account
 - Allows IAM users from a 3rd party AWS account to access this account and enforces use of External ID.
 - Select button
- Role for identity provider access

At the bottom are 'Cancel' and 'Next Step' buttons.

Po wybraniu opcji *Role for cross-account access* mamy dwie możliwości. Chcesz umożliwić dostęp do chmury niezależnej usłudze, a więc kliknij przycisk *Select* znajdujący się obok opcji *Provide access between your AWS account and a 3rd party AWS account*.

Właśnie przeszedłeś do drugiego etapu. Wprowadź identyfikator usługi zewnętrznej w polu *Account ID*. W tym przypadku identyfikatorem jest kod 689684103426. W polu *External ID* (identyfikator zewnętrzny) wprowadź swój identyfikator Docker ID. Pozostaw niezaznaczoną opcję *Require MFA* (wymagaj uwierzytelniania wielostopniowego):

Enter the ID of the 3rd party AWS account whose IAM users will be able to access this account. Enter the external ID provided by the 3rd party. For details, see [About the External ID](#).

Step 1 : Select role type

Step 2 : Establish trust

Step 3 : Attach policy

Step 4 : Set role name and review

Account ID: 689684103426

External ID: russmckendrick

Require MFA:

Cancel Previous Next Step

Po wypełnieniu formularza kliknij przycisk *Next Step* (kolejny krok). Teraz możesz dołączyć utworzoną wcześniej regułę. W tym celu w polu *Filter* (filtr) wpisz docker cloud lub początek innej nazwy, którą nadano podczas tworzenia zaprezentowanej wcześniej reguły, i zaznacz pole wyboru obok znalezionej nazwy utworzonej wcześniej reguły (w moim przypadku była to reguła *dockercloud-policy*):

Attach Policy

Select one or more policies to attach. Each role can have up to 10 policies attached.

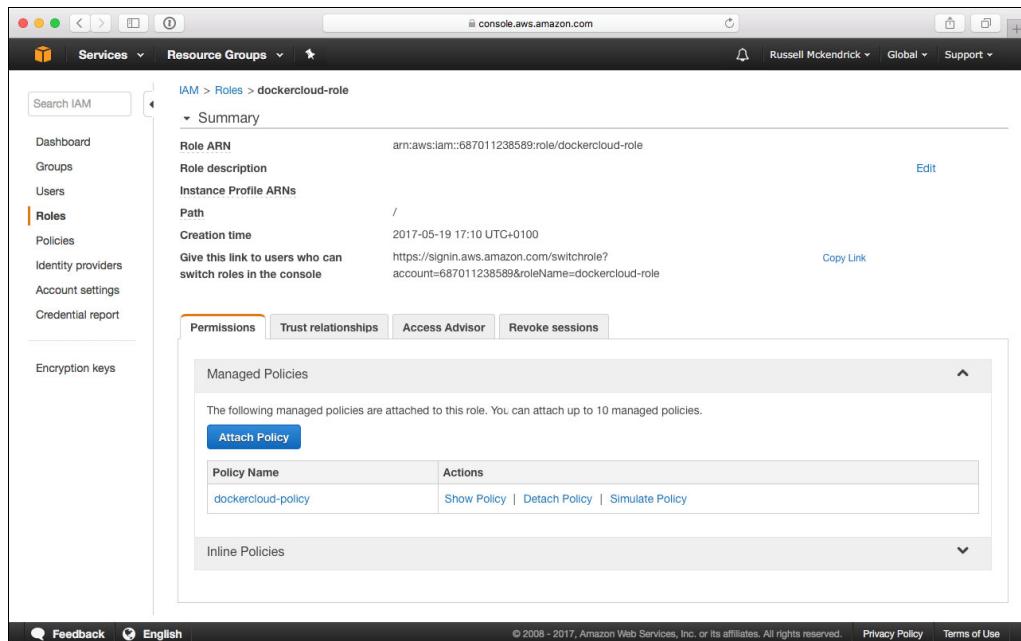
Filter: Policy Type ▾ docker.cloud Showing 1 results

| | Policy Name | Attached Entities | Creation Time | Edited Time |
|-------------------------------------|--------------------|-------------------|---------------------------|---------------------------|
| <input checked="" type="checkbox"/> | dockercloud-policy | 0 | 2017-05-19 17:08 UTC+0100 | 2017-05-19 17:08 UTC+0100 |

Cancel Previous Next Step

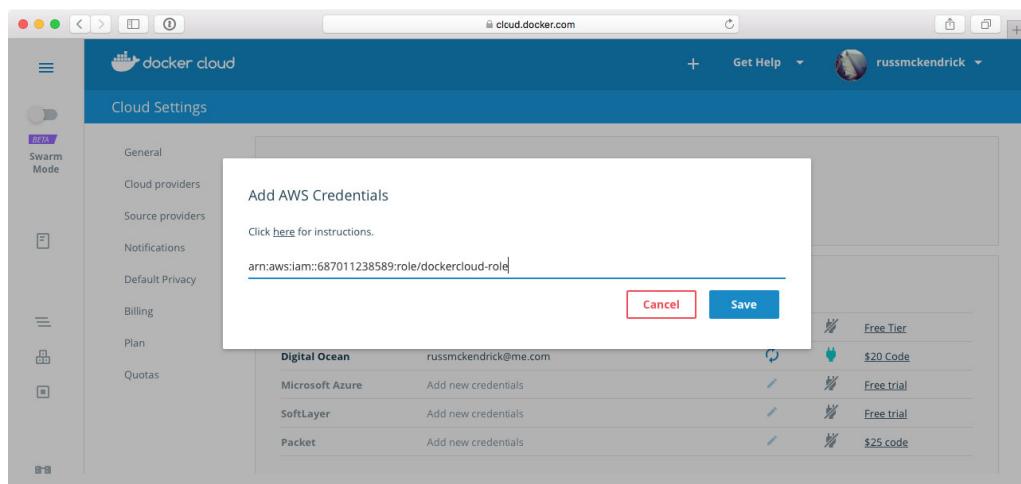
Kliknij przycisk *Next Step*, aby przejść do ostatniego kroku. Polega on na nadaniu tworzonej roli nazwy. W polu *Role Name* (nazwa roli) wpisz dockercloud-role. Sprawdź poprawność pozostałych danych wyświetlanych na tej stronie. Zwróć uwagę między innymi na numer konta (w polu tym powinien widnieć numer 689684103426), a następnie kliknij przycisk *Create role* (utwórz rolę).

Kliknięcie to spowoduje utworzenie roli Docker Cloud i przeniesie Cię z powrotem do listy ról. Zanim wrócisz do narzędzia Docker Cloud, musisz jeszcze ustalić nazwę ARN utworzonej przed chwilą roli. W tym celu kliknij nazwę roli widoczną na liście ról i skopiuj zawartość pola *Role ARN* (nazwa ARN roli):



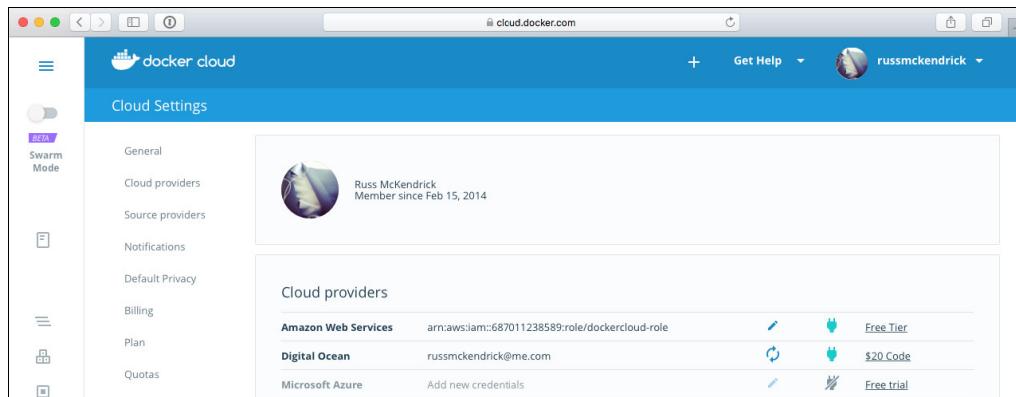
The screenshot shows the AWS IAM Roles page. On the left, there's a sidebar with options like Dashboard, Groups, Users, Roles (which is selected), Policies, Identity providers, Account settings, Credential report, and Encryption keys. The main area shows a role named 'dockercloud-role'. The 'Summary' tab is active, displaying the ARN: arn:aws:iam::687011238589:role/dockercloud-role. Below it, there are fields for Role description, Instance Profile ARNs, Path (/), Creation time (2017-05-19 17:10 UTC+0100), and a link to switch roles. A 'Copy Link' button is also present. At the bottom, there are tabs for Permissions, Trust relationships, Access Advisor, and Revoke sessions. Under the Permissions tab, there's a section for Managed Policies with a table showing one policy named 'dockercloud-policy' with actions for Show Policy, Detach Policy, and Simulate Policy.

Dysponujesz nazwą ARN roli, a więc wróć do usługi Docker Cloud i kliknij ikonę wtyczki znajdującą się obok *Amazon Web Services*. Na ekranie pojawi się prośba o podanie nazwy roli ARN. Wklej skopiowaną wcześniej nazwę i kliknij przycisk *Save*:



The screenshot shows the Docker Cloud Settings page. On the left, there's a sidebar with Swarm Mode (Beta), General, Cloud providers, Source providers, Notifications, Default Privacy, Billing, Plan, and Quotas. The main area shows a table of cloud providers: Digital Ocean, Microsoft Azure, SoftLayer, and Packet. A modal dialog box titled 'Add AWS Credentials' is open, containing a text input field with the value 'arn:aws:iam::687011238589:role/dockercloud-role'. There are 'Cancel' and 'Save' buttons at the bottom of the dialog. The background shows the list of providers with their respective status and pricing information.

Teraz Twoje konto Docker Cloud powinno być powiązane z chmurą AWS, a strona *Cloud Settings* (ustawienia chmury) powinna wyglądać tak:



Uruchamianie węzłów

Po połączeniu dwóch chmur możemy rozpocząć uruchamianie węzłów. Zaczniemy od utworzenia w chmurze DigitalOcean klastra składającego się z dwóch węzłów. Aby to zrobić, musimy kliknąć ikonę *Node Clusters* (klastry węzłów) znajdującą się w menu wyświetlonym po lewej stronie panelu Docker Cloud, a następnie kliknąć przycisk *Create*. Część formularza wypełnia się automatycznie, a więc nie martw się, jeżeli nie widzisz wszystkich poniższych opcji. Utwórzmy klaster o nazwie *Testing* (testowanie) o następujących parametrach:

- *NAME* (nazwa): *Testing*;
- *LABELS* (etykiet): pozostaw puste;
- *PROVIDER* (dostawca): *Digital Ocean*;
- *REGION*: wybierz najbliższy;
- *TYPE/SIZE* (typ i rozmiar): *1GB [1 CPUs, 1 GB RAM]*;
- *DISK SIZE* (przestrzeń dyskowa): pole to zostanie wypełnione automatycznie po uzupełnieniu pola *TYPE/SIZE*;
- *Number of nodes* (liczba węzłów): *2*.

Oto wypełniony formularz (patrz pierwszy zrzut na następnej stronie).

Pamiętaj, że wykonywanie kolejnych opisywanych przeze mnie czynności wiąże się z ponoszeniem kosztów za korzystanie z usług DigitalOcean i Docker Cloud. Jeżeli nie chcesz zwiększać kosztów użytkowania własnych kont, to możesz śledzić kolejne operacje na wykonanych przeze mnie zrzutach.

The screenshot shows the Docker Cloud Node Clusters / Wizard interface. On the left, there's a sidebar with icons for Swarm Mode, a list, and a file. The main area has a blue header with the Docker logo and 'Get Help'. Below the header, it says 'Node Clusters / Wizard'. A left panel contains fields for 'NAME' (Testing), 'LABELS' (Select...), 'PROVIDER' (Digital Ocean), 'REGION' (London 1), 'TYPE/SIZE' (1GB [1 CPUs, 1 GB RAM]), and 'DISK SIZE' (30 GB). To the right is a 'Number of nodes' section with a slider set to 2 and a dropdown menu showing '2 nodes'. At the bottom right is a large blue button labeled 'Launch node cluster'.

Kliknięcie przycisku *Launch node cluster* spowoduje uruchomienie klastra. W koncu DigitalOcean pojawią się dwa uruchomione przed chwilą apety (Droplets):

The screenshot shows the DigitalOcean Droplets list. The top navigation bar includes 'Droplets', 'Images', 'Networking', 'Monitoring', 'API', and 'Support'. There's a green 'Create Droplet' button and a user profile icon. The main area is titled 'Droplets' with a search bar 'Search by Droplet name'. Below is a table with columns: Name, IP Address, Created, and Tags. Two entries are listed:

| Name | IP Address | Created | Tags |
|---|---------------|--------------|----------------------|
| ffe6da2c-2fd4-4973-b0a3-4cad348b9136.n... | 46.101.50.207 | 1 minute ago | More |
| c389407d-e213-4384-91fc-8d46d58d9310.n... | 46.101.48.225 | 1 minute ago | More |

Po upływie kilku minut Twój klaster zyska status *DEPLOYED* (wdrożony):

The screenshot shows the Docker Cloud Node Clusters list. The top navigation bar includes 'Actions', 'Bring your own node', and 'Create'. The main area lists a single cluster named 'Testing' with a status of 'DEPLOYED'. It shows '2' nodes, 'London 1 • 1GB', and was created '3 minutes ago'. There's a small icon next to the cluster entry.

Kliknięcie nazwy klastra spowoduje wyświetlenie szczegółowych informacji na jego temat:

The screenshot shows the Docker Cloud web interface. On the left, there's a sidebar with icons for Swarm Mode, Nodes, and Settings. The main area has a header 'Node Clusters' with tabs 'General' (selected) and 'Timeline'. Below this, there's a cluster card for 'TESTING' in 'London 1' region. The card displays the following details:

- CPU: 1
- MEMORY: 1024 MB
- DISK SPACE: 30 GB

A timestamp 'a few seconds ago' is shown at the bottom. To the right of the cluster card is a 'Scale' slider and two buttons: 'Terminate' (red) and 'Deploy' (green). A user profile 'russmckendrick' is visible in the top right.

Kliknięcie któregoś z dwóch węzłów spowoduje wyświetlenie jego parametrów:

This screenshot shows the Docker Cloud interface with the 'Nodes' section selected in the sidebar. It displays a single node entry for 'Testing' with the ID 'c389407d-e213-4384-91fc-8d4...'. The 'General' tab is selected. The node details are as follows:

- IP: 46.101.48.225
- CLOUD PROVIDER: DIGITALOCEAN
- REGION: lon1
- NODE TYPE: 1GB
- MEMORY: 1024 MB
- DISK SPACE: 30 GB

To the right of the node card, there's a 'DOCKER INFO' section showing:

- Version 1.11.2-cs5
- Graph driver aufs
- Exec driver

Below the node card is an 'Endpoints' section with the message 'No endpoint defined'. A 'Terminate' button is located in the top right corner of the node card area.

Wersje węzłów Dockera są inne od tych, z których korzystaliśmy wcześniej. Teraz korzystamy z silnika Docker Engine w wersji **CS — Commercially Supported** (wsparcie komercyjne). Ma on takie same funkcje jak otwarta wersja Dockera 1.11.2, ale zawiera dodatkowo poprawki związane z bezpieczeństwem i problemami znymi z innych wersji.

Uruchamianie stosu

Dysponujemy już klastrem z kilkoma aktywnymi węzłami, a więc możemy przystąpić do uruchamiania kontenerów. Będziemy ponownie uruchamiać aplikację o nazwie *cluster*. Zrobimy to za pomocą pliku stosu podobnego do tego, który został przedstawiony w rozdziale 8. „Portainer”. Plik, z którego będziemy korzystać, ma teraz nieco inną składnię — ma współpracować z innym narzędziem.

Oto plik definiujący stos naszej aplikacji:

```
1b:  
  image: dockercloud/haproxy  
  autorestart: always  
  links:  
    - web  
  ports:  
    - "80:80"  
  roles:  
    - global  
  web:  
    image: russmckendrick/cluster  
    autorestart: always  
    deployment_strategy: high_availability  
    target_num_containers: 4
```

Tworzymy dwie usługi: usługę równoważący obciążenie i aplikację sieciową (*web*). Mechanizm równoważenia obciążenia jest oparty na obrazie Docker Cloud HAProxy, a aplikacja sieciowa jest oparta na naszym własnym obrazie pobieranym z repozytorium Docker Hub. W celu uruchomienia stosu kliknij ikonę *Stacks* (stosy) znajdującą się w menu wyświetlanym po lewej stronie, a następnie wprowadź zaprezentowany wcześniej kod tworzący stos. Pamiętaj o konieczności zachowania wcięć i spacji — bez tego Docker Cloud nie będzie w stanie zinterpretować poprawnie wprowadzonego kodu. Tworzonymu stosowi nadałem nazwę *ClusterApp*:

```

ClusterApp

1 - lb:
2   image: dockercloud/haproxy
3   autorestart: always
4 - links:
5   - web
6 - ports:
7   - "80:80"
8 - roles:
9   - global
10 - web:
11   image: russmckendrick/cluster
12   autorestart: always
13   deployment_strategy: high_availability
14   target_num_containers: 4

```

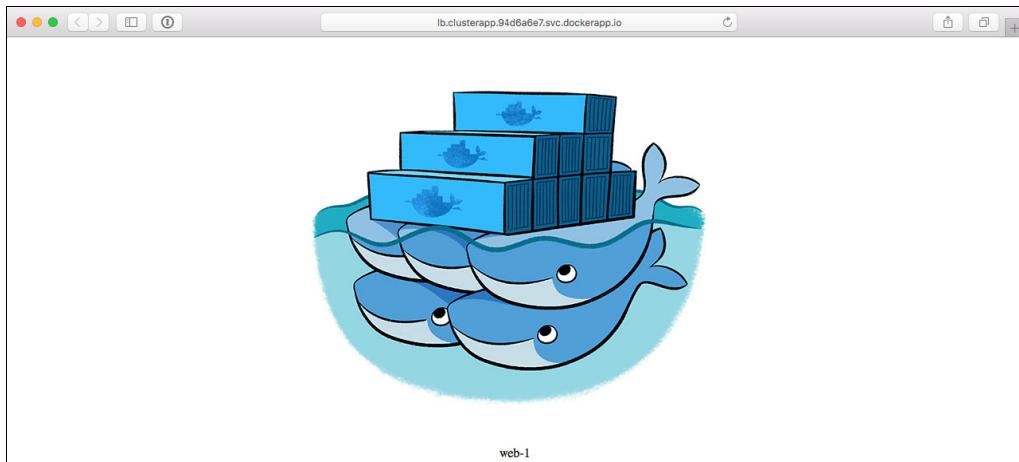
Use 2 spaces for indentation. Not sure what a Stackfile is? [Click here to learn more](#).

Create & Deploy

Po wprowadzeniu kodu opisującego tworzony stos kliknij przycisk *Create & Deploy* (utwórz i wdróż). Zostaniesz przeniesiony bezpośrednio do stosu. Na ekranie pojawią się informacje o usługach i węźle końcowym (sekcja *Endpoints*):

| CLUSTERAPP | | |
|-------------------|-------------------|---|
| RUNNING | | |
| web | a few seconds ago | lb |
| Endpoints | | |
| Service Endpoints | | |
| lb | tcp/80 | http://lb.clusterapp.94d6a6e7.svc.dockerapp.io:80 |

Węzeł końcowy jest zdefiniowany za pomocą pełnego adresu URL w domenie *dockerapp.io*. Mojemu węzlowi końcowemu przypisano adres *http://lb.clusterapp.94d6a6e7.svc.dockerapp.io/* (adres ten nie jest już aktywny). Po wprowadzeniu adresu swojego węzła końcowego do przeglądarki powinieneś zobaczyć aktywną aplikację:



Po kliknięciu ikony *Services* (usługi) pojawi się lista dwóch usług:

A screenshot of the Docker Cloud Services interface. The left sidebar shows "Swarm Mode" is selected. The main area is titled "Services". It lists two services: "web" and "lb". The "web" service has 4 instances of "ClusterApp" running on "cluster:latest", and the "lb" service has 1 instance of "ClusterApp" running on "haproxy:latest". Both services are marked as "RUNNING". There are "Actions" and "Create" buttons at the top right of the service list.

Strona ta umożliwia zatrzymanie (*Stop*), uruchomienie (*Start*), ponowne wdrożenie (*Redeploy*) i zamknięcie (*Terminate*) usług. Kliknięcie nazwy usługi spowoduje wyświetlenie dodatkowych informacji dotyczących jej wdrożenia, a także listy aktywnych kontenerów tworzących usługę:

The screenshot shows the Docker Cloud Services interface. On the left, there's a sidebar with icons for Swarm Mode, a list, a file, a network, and a terminal. The main area has a blue header bar with the Docker logo and the text "cloud.docker.com". Below the header, there's a navigation bar with tabs: "General" (which is selected), "Logs", and "Timeline". To the right of the tabs are buttons for "Edit", "Actions", and "Stop". The main content area displays a single service entry:

- Stack Name:** ClusterApp
- Image Tag:** russmckendrick/cluster:latest
- Run Command:** (disabled)
- Sequential Deployment:** OFF
- Deployment Strategy:** HIGH_AVAILABILITY
- Privileged Mode:** OFF
- Autorestart:** ALWAYS
- Autoredeploy:** OFF
- Autodestroy:** OFF
- Network:** bridge
- Ports:** (empty)

Below the service details, it says "WEB" and "RUNNING", with a timestamp of "23 minutes ago".

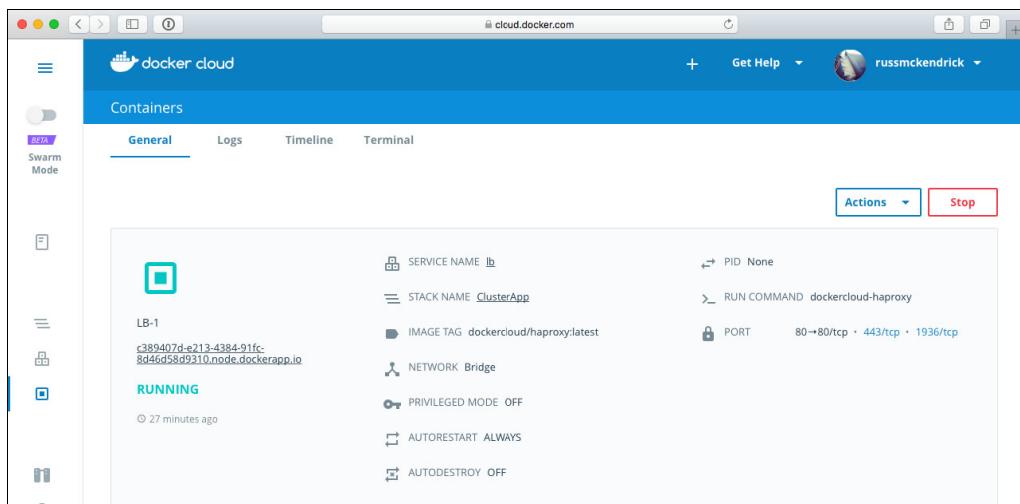
Kliknięcie opcji *Containers* (kontenery) znajdującej się w menu wyświetlanym po lewej stronie spowoduje wyświetlenie listy kontenerów aktywnych w koncie Docker Cloud:

The screenshot shows the Docker Cloud Containers interface. The left sidebar is identical to the Services interface. The main area has a blue header bar with the Docker logo and the text "cloud.docker.com". Below the header, there's a search bar with "Select All" and "Filter by name...". To the right of the search bar is a "Actions" button. The main content area lists five containers:

| Name | Type | Image | Created | Actions |
|-------|------|----------------|----------------|------------------------|
| web-1 | web | cluster:latest | 25 minutes ago | [stop, start, inspect] |
| web-2 | web | cluster:latest | 25 minutes ago | [stop, start, inspect] |
| web-3 | web | cluster:latest | 25 minutes ago | [stop, start, inspect] |
| web-4 | web | cluster:latest | 25 minutes ago | [stop, start, inspect] |
| lb-1 | lb | haproxy:latest | 25 minutes ago | [stop, start, inspect] |

A small number "1" is visible at the bottom center of the container list.

Z wyświetlonej listy wynika, że mamy pięć aktywnych kontenerów: cztery z nich tworzą aplikację sieciową, a piąty zapewnia mechanizm równoważenia obciążzeń. Po kliknięciu nazwy kontenera na ekranie pojawiają się informacje na jego temat:



Ponadto możesz wyświetlić zawartość dziennika wybranego kontenera. Po kliknięciu przycisku *Logs* (dziennik) na ekranie będą wyświetlane wpisy dziennika w czasie rzeczywistym, tak jak w przypadku uruchomienia polecenia `docker container logs` w wierszu poleceń. Kliknięcie przycisku *Terminal* spowoduje otwarcie w przeglądarce interaktywnego Terminala, który umożliwia uzyskanie dostępu do wiersza poleceń samego kontenera.

Teraz już wiesz, jak uruchomić klaster w chmurze Amazon Web Services.

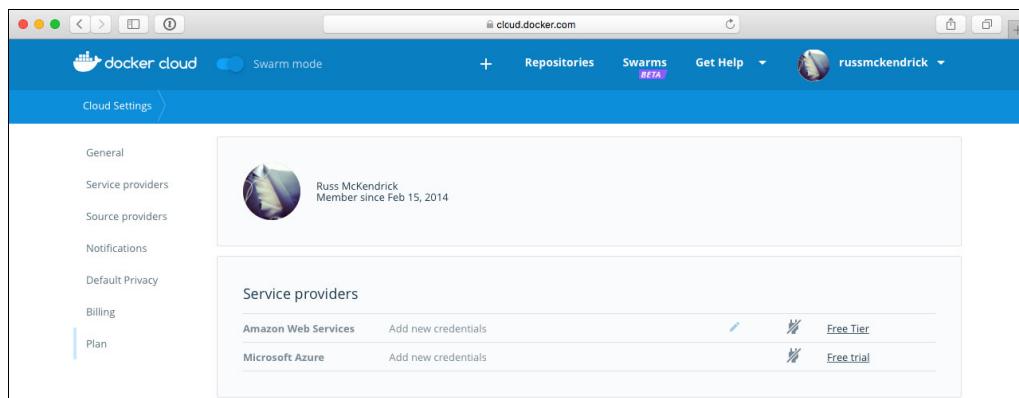
Jeżeli wykonywałeś opisywane przeze mnie czynności, to teraz jest dobra chwila na to, aby zakończyć pracę stosów i klastrów, dzięki czemu nie poniesiesz zbędnych wydatków.

Tryb Swarm

Z pewnością zauważyleś obecność przycisku *Swarm mode (beta)* (tryb Swarm) w górnej części menu znajdującego się po lewej stronie. Kliknięcie go spowoduje przejście do wersji beta nowej wersji usługi Docker Cloud. Wersja ta nie jest środowiskiem, do którego po prostu dodano kilka nowych funkcji — jest to zupełnie nowe środowisko, które całkowicie zastępuje obecną wersję Docker Cloud.

Po przejściu do sekcji *Cloud Settings* (przycisk ten znajduje się teraz w rozwijanym menu wyświetlonym u góry ekranu, obok nazwy użytkownika) zobaczyś listę dwóch dostawców usług (*Service providers*). Ponadto nie znajdziesz tam identyfikatora ARN utworzonej wcześniej roli (patrz zrzut na następnej stronie).

Niestety korzystanie ze skonfigurowanej wcześniej roli jest niemożliwe, ponieważ **tryb Swarm** wymaga wielu dodatkowych pozwoleń. Zamiast przeprowadzać proces dodawania nowej roli dla trybu Swarm, opiszemy uruchamianie trybu Swarm przy użyciu Dockera dla usług Amazon Web Services.



Nie będziemy opisywać trybu Swarm usługi Docker Cloud. Jeżeli chcesz uzyskać informacje na temat tworzenia identyfikatora roli i korzystania z usług, to powinieneś zatrzymać się na stronie <https://docs.docker.com/docker-cloud/cloud-swarm/>.

Przed przejściem do kolejnego rozdziału sprawdź usunięcie wszystkich zbędnych stosów i klastrów uruchomionych za pomocą usługi Docker Cloud.

Docker dla Amazon Web Services

Docker dla Amazon Web Services (Docker for AWS) jest szablonem Amazon CloudFormation utworzonym przez twórców Dockera, który ma ułatwiać uruchamianie klastrów w trybie Docker Swarm w chmurze AWS przy zachowaniu dobrych praktyk obsługi platformy Docker.

Amazon CloudFormation to usługa firmy Amazon umożliwiająca definiowanie infrastruktury za pomocą pliku szablonu, który może być udostępniany lub obsługiwany przez narzędzia przeznaczone do kontroli wersji. Więcej informacji na temat tej usługi znajdziesz na stronie <https://aws.amazon.com/cloudformation/>.

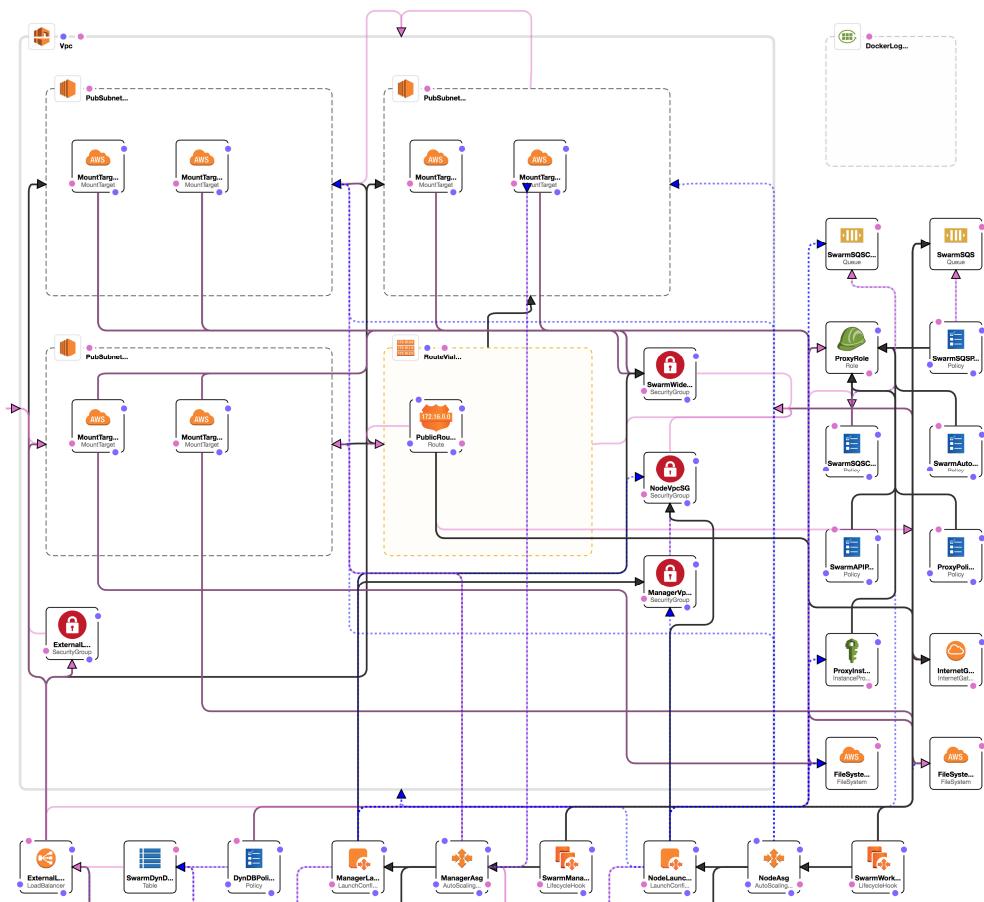
Przed uruchomieniem szablonu *Docker for AWS* musimy wykonać tylko jedną czynność — powinniśmy upewnić się, że do naszego konta przypisana jest usługa SSH działająca w regionie, w którym będziemy uruchamiać klaszter. W tym celu musimy zalogować się do konsoli AWS znajdującej się na stronie <https://console.aws.amazon.com/> (jeżeli pracujesz w firmie, która korzysta z własnej strony logowania, to skorzystaj z niej). Po zalogowaniu się przejdź do menu *Service* (usługa) znajdującego się w lewym górnym rogu strony i znajdź usługę *EC2*.

W celu upewnienia się, że jesteś we właściwym regionie, skorzystaj z przełącznika regionu znajdującego się w prawej górnej części strony (pomiędzy nazwą użytkownika a menu pomocy — *Support*). Po wybraniu właściwego regionu kliknij przycisk *Key Pairs* (klucze par) znajdujący się w menu *Network & Security* (sieć i bezpieczeństwo) wyświetlonym po lewej stronie ekranu.

Po wejściu na stronę *Key Pairs* zobaczysz listę bieżących par kluczy. Jeżeli lista jest pusta lub nie masz dostępu do żadnych kluczy, kliknij przycisk *Create Key Pair* (utwórz parę kluczy) lub *Import Key Pair* (importuj parę kluczy) i wykonuj polecenia wyświetlane na ekranie.

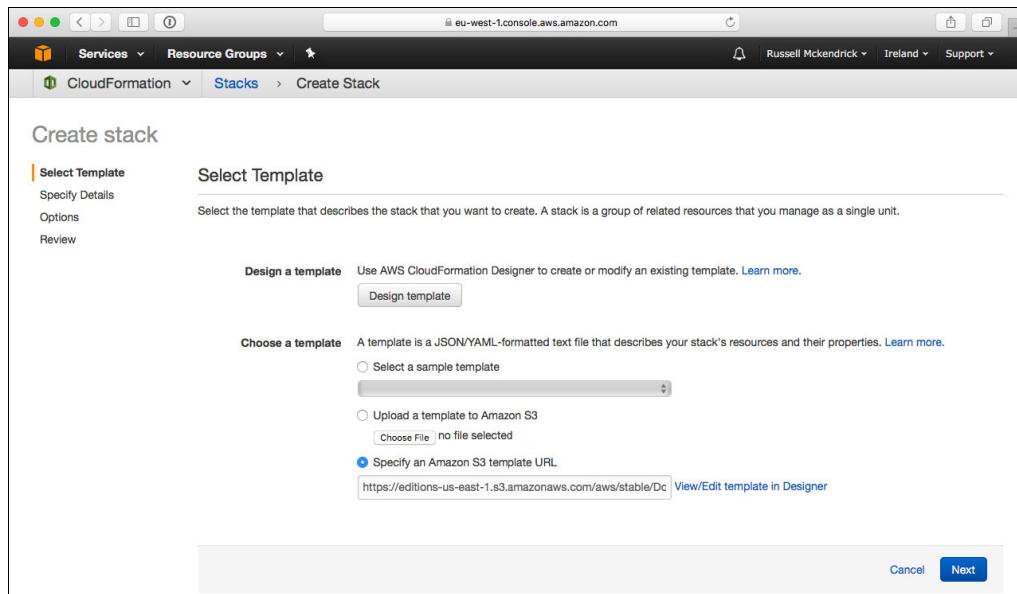
Szablon *Docker for AWS* znajduje się w sklepie Docker Store pod adresem <https://store.docker.com/editions/community/docker-ce-aws>. Możesz wybrać spośród dwóch wersji: *stable* (stabilna) i *Edge* (wersja eksperymentalna zawierająca funkcje, które zostaną wprowadzone w wersji stabilnej po dopracowaniu). Będziemy korzystać z wersji stabilnej. Kliknij jej przycisk, a zostaniesz przeniesiony od razu do usługi CloudFormation, w której załadowany będzie szablon Dockera.

Surowy szablon możesz obejrzeć na stronie <https://editions-us-east-1.s3.amazonaws.com/aws/stable/Docker.tmpl>. Ponadto w narzędziu projektowym CloudFormation możesz dokonać jego wizualizacji. Jak wynika z poniższej wizualizacji, uruchomienie klastra wymaga wykonania wielu operacji:



Na szczęście nie musisz przejmować się żadną z tych komplikacji. Docker wykona wszystkie skomplikowane operacje za Ciebie.

W kolejnym kroku uruchamiania klastra nie musisz niczego zmieniać — na stronie *Select Template* (wybierz szablon) wystarczy kliknąć przycisk *Next* (dalej):



Na stronie *Specify Details* musimy określić szczegółowe parametry klastra. Poza parametrem *SSH Key* (klucz SSH) pozostawimy tutaj same domyślne wartości:

- *Stack name* (nazwa stosu): Docker;
- *Number of Swarm managers?* (liczba menedżerów Swarm): 3;
- *Number of Swarm worker nodes?* (liczba węzłów roboczych Swarm): 5;
- *Which SSH key to use?* (używany klucz SSH): wybierz swój klucz z listy;
- *Enable daily resource cleanup?* (włącz codzienne czyszczenie zasobów): No (nie);
- *Use Cloudwatch for container logging?* (używaj narzędzia Cloudwatch do obsługi dzienników kontenerów): Yes (tak);
- *Swarm manager instance type?* (typ instancji menedżera Swarm): t2.micro;
- *Manager ephemeral storage volume size?* (rozmiar wolumenu magazynu ulotnego menedżera): 20;
- *Manager ephemeral storage volume type* (typ wolumenu magazynu ulotnego menedżera): standard (standardowy);
- *Agent worker instance type?* (typ instancji podmiotu roboczego): t2.micro;

- *Worker ephemeral storage volume size?* (rozmiar wolumenu magazynu ulotnego podmiotu roboczego): 20;
- *Worker ephemeral storage volume type* (typ wolumenu magazynu ulotnego podmiotu roboczego): *standard* (standardowy).

Po sprawdzeniu wszystkich parametrów kliknij przycisk *Next*. W kolejnym kroku możesz niczego nie zmieniać i kliknąć przycisk *Next* w celu przejścia do strony podsumowującej. Na stronie tej znajdziesz odwołanie kierujące do szacunków dotyczących kosztu utrzymania zdefiniowanego klastra:

The screenshot shows the AWS Simple Monthly Calculator interface. At the top, it says "FREE USAGE TIER: New Customers get free usage tier for first 12 months". Below that, there's a "Services" dropdown set to "Amazon EC2" and a table titled "Compute: Amazon EC2 Instances". The table lists two instances: "ManagerAsg" and "NodeAsg", both running on t2.micro instances 24 hours a day. The total monthly cost is \$113.46. To the right, there's a sidebar with "Common Customer Samples" including "AWS Elastic Beanstalk Default", "Marketing Web Site", and "Large Web Application (All On-Demand)". At the bottom, there's a note about Inbound Data Transfer being free.

Jak widzisz, w przypadku mojej konfiguracji szacunkowy koszt utrzymania klastra wynosi 113,46 dolara. Przed uruchomieniem klastra należy naznaczyć pole *I acknowledge that AWS CloudFormation might create IAM resources* (jestem świadomym tego, że AWS CloudFormation może tworzyć zasoby IAM) i kliknąć przycisk *Create* (utwórz). Uruchamianie klastra może trochę potrwać. W celu sprawdzenia statusu uruchamianego klastra wybierz stos *CloudFormation* w zakładce *Events* (zdarzenia) konsoli AWS:

The screenshot shows the AWS CloudFormation Events tab. It displays a table of events for a stack named "Docker". The table includes columns for Stack Name, Created Time, Status, and Description. One event is shown: "CREATE_IN_PROGRESS" for "Docker" at 2017-05-21 12:47:00 UTC+0100, with the description "Docker for AWS 17.03.1-ce (aws2)". Below the table, there's a list of events with detailed information about each resource creation.

| Stack Name | Created Time | Status | Description |
|------------|------------------------------|--------------------|----------------------------------|
| Docker | 2017-05-21 12:47:00 UTC+0100 | CREATE_IN_PROGRESS | Docker for AWS 17.03.1-ce (aws2) |

| Date | Status | Type | Logical ID | Status reason |
|-------------------|--------------------|---------------------|-----------------|-----------------------------|
| 2017-05-21 | | | | |
| 12:47:08 UTC+0100 | CREATE_COMPLETE | AWS::SQS::Queue | SwarmSQSCleanup | |
| 12:47:08 UTC+0100 | CREATE_COMPLETE | AWS::SQS::Queue | SwarmSQS | |
| 12:47:08 UTC+0100 | CREATE_COMPLETE | AWS::Logs::LogGroup | DockerLogGroup | |
| 12:47:07 UTC+0100 | CREATE_IN_PROGRESS | AWS::SQS::Queue | SwarmSQSCleanup | Resource creation initiated |
| 12:47:07 UTC+0100 | CREATE_IN_PROGRESS | AWS::SQS::Queue | SwarmSQS | Resource creation initiated |
| 12:47:07 UTC+0100 | CREATE_IN_PROGRESS | AWS::Logs::LogGroup | DockerLogGroup | Resource creation initiated |
| 12:47:07 UTC+0100 | CREATE_IN_PROGRESS | AWS::EC2::VPC | Vpc | Resource creation initiated |
| 12:47:07 UTC+0100 | CREATE_IN_PROGRESS | AWS::IAM::Role | ProxyRole | Resource creation initiated |
| 12:47:07 UTC+0100 | CREATE_IN_PROGRESS | AWS::SQS::Queue | SwarmSQS | Resource creation initiated |

Po upływie około 10 minut status stosu powinien zmienić się z *CREATE_IN_PROGRESS* (w trakcie tworzenia) na *CREATE_COMPLETE* (tworzenie zakończone). Gdy tak się stanie, kliknij zakładkę *Outputs*. Znajdziesz tam listę adresów URL i odnośników:

| Key | Value | Description | Export Name |
|-------------------------|---|--|-------------|
| DefaultDNSTarget | Docker-ELB-1700993975.eu-west-1.elb.amazonaws.com | Use this name to update your DNS records | |
| ZoneAvailabilityComment | This region has at least 3 Availability Zones (AZ). This is ideal to ensure a fully functional Swarm in case you lose an AZ. | Availability Zones Comment | |
| Managers | https://eu-west-1.console.aws.amazon.com/ec2/v2/home?region=eu-west-1#instances:tag:aws:autoscaling:groupName=Docker-ManagerAsg-VSB9VT1F7IKL;sort=desc:dnsName | You can see the manager nodes associate... | |

W celu zalogowania się do klastra Swarm należy kliknąć link znajdujący się obok menedżerów. Spowoduje to wyświetlenie listy instancji EC2 (węzłów menedżera). Wybierz jedną z instancji, a następnie zapisz jej publiczny adres IP. Skorzystaj z Terminala i protokołu SSH w celu nawiązania połączenia z węzłem (zaloguj się jako użytkownik docker). Oto polecenia, które uruchomiłem w celu zalogowania się i wyświetlenia listy wszystkich moich węzłów:

```
$ ssh docker@54.246.218.236
$ docker node ls
```

W przypadku pobrania klucza SSH z konsoli AWS podczas wykonywania operacji dodawania klucza musisz zmodyfikować powyższe polecenie i dodać do niego ścieżkę pobranego klucza. Oto przykład zmodyfikowanego polecenia: `ssh -i /folder/klucza/private.key docker@54.246.218.23`.

Oto okno Terminala, w którym uruchomiono zaprezentowane wcześniej polecenia w celu zalogowania się i pobrania listy wszystkich węzłów:

```
russ in ~
$ ssh docker@54.246.218.236
The authenticity of host '54.246.218.236 (54.246.218.236)' can't be established.
ECDSA key fingerprint is SHA256:QSpq4Uc/AJT2yVe55dh5QEMGjs3Y1HzInFxTUtnMrWk.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.246.218.236' (ECDSA) to the list of known hosts.
Welcome to Docker!
~ $ docker node ls
ID          HOSTNAME   STATUS  AVAILABILITY  MANAGER STATUS
d8smypcnt0o5osl80ncvvr1p  ip-172-31-1-133.eu-west-1.compute.internal  Ready  Active
ite13odzhudtw2p1knr0zkztm ip-172-31-22-67.eu-west-1.compute.internal  Ready  Active
jw1iqvp6f6lku823qoiefkqy * ip-172-31-44-79.eu-west-1.compute.internal  Ready  Active
Leader
nd2g2pk1cxpb91w9valonyc7o ip-172-31-26-81.eu-west-1.compute.internal  Ready  Active
s1dru2ewgtad5pelh8k7dk4ds ip-172-31-35-46.eu-west-1.compute.internal  Ready  Active
vgch3a75rmmh9wftyu59xatgk ip-172-31-6-9.eu-west-1.compute.internal  Ready  Active
Reachable
w5zqd02771xz1z40pn8fkw30u5 ip-172-31-36-40.eu-west-1.compute.internal  Ready  Active
zj85d1b78lpppzqzgwzg7bt84 ip-172-31-1-229.eu-west-1.compute.internal  Ready  Active
~ $
```

Od teraz klaster można traktować tak, jakby był standardowym klastrem Docker Swarm. Możesz uruchomić i skalować usługę klastra:

```
$ docker service create --name cluster --constraint "node.role == worker" -  
→p:80:80/tcp russmckendrick/cluster  
$ docker service scale cluster=6  
$ docker service ls  
$ docker service inspect --pretty cluster
```

Po uruchomieniu usługi możesz sprawdzić działanie aplikacji za pomocą adresu URL oznaczonego identyfikatorem *DefaultDNSTarget*. Pod adresem tym znajduje się mechanizm równoważenia obciążenia Amazon Elastic, który rozdziela ruch pomiędzy czterema skonfigurowanymi wcześniej węzłami.

Po skończeniu pracy nad klastrem wróć na stronę *CloudFormation* znajdująca się w konsoli AWS, a następnie wybierz opcję *Delete Stack* (usuń stos) z rozwijanego menu *Actions* (akcje). Spowoduje to usunięcie utworzonego klastra i zaprzestanie naliczania opłat za jego funkcjonowanie.

Istnieje również wersja Dockera przystosowana do współpracy z chmurą Azure. Działa ona podobnie do Dockera przystosowanego do współpracy z chmurą Amazon Web Services, ale wymaga nieco dłuższej konfiguracji. Więcej informacji na ten temat znajdziesz na stronie <https://store.docker.com/editions/community/docker-ce-azure>.

Podsumowanie

W tym rozdziale przyjrzaliśmy się trzem graficznym interfejsom Dockera. Narzędzia te mogą być używane do wykonywania operacji związanych z niemalże każdym elementem infrastruktury Dockera: od środowisk hosta do znajdujących się w nich obrazów i kontenerów. Umożliwiają one przeprowadzanie operacji skalowania, modyfikowania i usuwania.

W kolejnym rozdziale przyjrzymy się technikom zabezpieczania hosta Dockera, a także skanowania obrazów kontenerów.

Bezpieczeństwo platformy Docker

W tym rozdziale przyjrzymy się obecnie bardzo ważnemu zagadniению bezpieczeństwa platformy. Rozdział zostanie podzielony na pięć podrozdziałów, w których poruszymy następujące tematy:

- bezpieczeństwo kontenerów;
- polecenia Dockera;
- dobre praktyki;
- aplikacja Docker Bench Security;
- niezależne usługi zabezpieczające.

Bezpieczeństwo kontenerów

Po udostępnieniu pierwszej wersji Dockera wiele osób porównywało kontenery i maszyny wirtualne. Artykuły i wzmianki na ten temat pojawiały się w czasopismach branżowych, a także na forach dyskusyjnych i blogach. Gdy Docker był jeszcze w fazie wersji alfa i beta, użytkownicy traktowali go jak maszynę wirtualną. Wynikało to z braku innego punktu odniesienia. Kontenery Dockera traktowano po prostu jak miniaturowe maszyny wirtualne.

W kontenerach można włączyć obsługę protokołu SSH, uruchamiać wiele procesów, a nawet tworzyć ich obrazy poprzez uruchamianie poleceń instalujących stos oprogramowania, co zgodnie z treścią rozdziału 2. „Tworzenie obrazów kontenerów” jest uważane za złą praktykę.

Nie będziemy porównywać kontenerów i maszyn wirtualnych. Zamiast tego poruszymy zagadnienia, o których warto pamiętać podczas pracy z kontenerami (zagadnienia te nie dotyczą maszyn wirtualnych).

Zalety

Podczas uruchamiania kontenera Dockera silnik Docker Engine wykonuje wiele operacji w tle. Dwie z tych operacji konfigurują przestrzenie nazw i grupy sterujące cgroups. Co umożliwia te operacje? Docker dzięki konfiguracji przestrzeni nazw izoluje procesy poszczególnych kontenerów nie tylko od innych kontenerów, ale także od systemu hosta. Grupy cgroups zapewniają każdemu kontenerowi dostęp do zasobów takich jak procesor, pamięć i operacje dyskowe. Co ważne, dzięki nim żaden kontener nie może wykorzystać wszystkich dostępnych zasobów hosta.

Przypominamy, że możliwość uruchomienia aplikacji w sieci sterowanej za pomocą środowiska Docker umożliwia izolowanie kontenerów na poziomie aplikacji — dzięki temu mechanizmowi aplikacja A nie ma dostępu do warstwy sieci kontenerów aplikacji B.

Mechanizm izolacji ruchu sieciowego może działać w ramach hosta Dockera (domyślny sterownik sieci) lub w ramach wielu hostów (sterownik obsługi wielu hostów jest wbudowany w narzędzie Docker Swarm, ale istnieje również niezależny sterownik Weave Net).

Ostatnią cechą pokazującą przewagę Dockera nad typowymi maszynami wirtualnymi jest brak konieczności logowania się do kontenera. Docker robi wszystko, aby użytkownik nie musiał logować się do kontenera i ręcznie zarządzać uruchomionymi w nim procesami. Polecenia takie jak `docker container exec`, `docker container top`, `docker container logs` i `docker container stats` pozwalają na wykonywanie wszystkich operacji bez zbędnej konieczności odsłaniańego dostępu do usług.

Host Dockera

Podczas pracy z maszynami wirtualnymi można określić uprawnienia dostępu użytkowników do poszczególnych maszyn. Programiście można umożliwić tylko dostęp do wybranych maszyn wirtualnych, a administratorowi nadać uprawnienia dostępu do wszystkich środowisk (w tym również środowisk produkcyjnych). Większość narzędzi przeznaczonych do zarządzania maszynami wirtualnymi umożliwia nadawanie uprawnień dostępu do maszyn wirtualnych.

W przypadku Dockera każda osoba, która uzyska dostęp do silnika Docker Engine hosta za pomocą konta administratora lub poprzez dodanie konta użytkownika do grupy Docker w Linuksie, ma dostęp do każdego uruchomionego kontenera. Osoba taka może uruchamiać nowe kontenery, zatrzymywać pracę uruchomionych wcześniej kontenerów i kasować obrazy. Zachowaj ostrożność, dając komuś dostęp do silnika Docker Engine hosta, ponieważ znajdują się tam klucze do wszystkich kontenerów. W związku z tym zaleca się, aby oddzielać inne usługi od hosta Dockera (host powinien obsługiwać tylko Dockera).

Zaufane źródła obrazów

Podczas pracy z maszynami wirtualnymi zwykle tworzy się je samodzielnie od podstaw. Z powodu rozmiaru obrazu maszyny wirtualnej i nakładu pracy niezbędnego do jej uruchomienia zwykle nie warto pobierać obrazu maszyny utworzonego przez anonimowego użytkownika internetu. Z gotowych obrazów korzysta się tylko wtedy, gdy pochodzą one od zaufanych twórców oprogramowania.

W wyniku tworzenia maszyny wirtualnej od podstaw wiadomo, co się w niej znajduje.

Jedną z zalet Dockera jest prostota jego obsługi. Niestety prostota ta sprawia, że bardzo łatwo zapomnieć o podstawowej zasadzie bezpieczeństwa: czy wiesz, jaki kod jest uruchamiany wewnątrz kontenera?

W poprzednich rozdziałach wspominaliśmy o zagrożeniach związanych z pobieraniem obrazów z nieznanych źródeł. Wiesz już, że nie warto pobierać ani publikować obrazów, które nie zostały zdefiniowane przy użyciu plików Dockerfile, a ponadto do obrazów kontenerów udostępnianych w repozytorium Docker Hub nie powinno się dodawać opracowanego samodzielnie kodu.

Pisaliśmy już, że korzystanie z obrazów niewiadomego pochodzenia może być poważnym zagrożeniem dla Twojego środowiska pomimo naturalnych zabezpieczeń kontenerów (przestrzeni nazw, grup cgroup i izolacji sieci). Nawet poprawnie utworzony kontener z nieaktualnym oprogramowaniem może spowodować problemy związane z dostępnością aplikacji lub danych.

Polecenia Dockera

Przyjrzymy się poleceniom Dockera, które mogą pomóc zwiększyć bezpieczeństwo, a także umożliwić podgląd informacji o obrazach, których użycie rozważasz. Skupimy się na dwóch poleceniach.

Pierwsze z nich to docker container run. Dowiesz się, jak w pełni wykorzystać jego możliwości. W dalszej części tego podrozdziału przyjrzymy się poleceniu docker container diff. Z polecenia tego korzystaliśmy w poprzednim rozdziale, ale tym razem użyjemy go w celu sprawdzenia operacji przeprowadzonych na obrazie, który ma zostać użyty w naszej infrastrukturze.

Polecenie run

W przypadku polecenia docker run skupimy się na opcji umożliwiającej zmianę parametru dostępu do wszystkich plików znajdujących się w kontenerze na „tylko do odczytu” (nie będziemy modyfikować parametrów tylko wybranego folderu lub wolumenu). Opcja ta pozwala ograniczyć zakres strat wywołanych przez złośliwe oprogramowanie próbujące przejąć kontrolę nad podatnymi na to aplikacjami poprzez modyfikację ich plików binarnych.

Oto przykładowe polecenie uruchamiające kontener, którego zawartość jest przeznaczona tylko do odczytu:

```
$ docker container run -d --name mysql --read-only -v /var/lib/mysql -v /tmp -v  
/var/run/mysqld -e MYSQL_ROOT_PASSWORD=password mysql
```

Polecenie to uruchamia kontener MySQL i sprawia, że cała zawartość kontenera jest przeznaczona tylko do odczytu, ale nie dotyczy to następujących folderów:

- */var/lib/mysql*;
- */var/run/mysqld*;
- */tmp*.

Foldery te zostaną utworzone w formie trzech oddzielnych wolumenów zamontowanych z uprawnieniami do odczytu i zapisu. Jeżeli te wolumeny nie zostaną dodane, to kontener MySQL nie będzie mógł zostać uruchomiony, ponieważ uprawnienia do zapisu i odczytu danych są niezbędne do utworzenia pliku gniazda w folderze */var/run/mysql*, obsługi plików tymczasowych w folderze */tmp* i samych baz danych w folderze */var/lib/mysql*.

W innych folderach kontenera nie można przeprowadzać operacji zapisu danych. Próba uruchomienia następującego polecenia zakończy się niepowodzeniem:

```
$ docker container exec mysql touch /testowy_zapis_pliku
```

Zwrócony zostanie następujący komunikat:

```
touch: cannot touch '/testowy_zapis_pliku': Read-only file system
```

Rozwiązanie to jest szczególnie przydatne wtedy, gdy chcesz określić miejsca, w których kontenery mogą zapisywać dane, i zdefiniować miejsca, w których dane nie mogą być modyfikowane. Korzystając z tej funkcji, należy zachować ostrożność, ponieważ zablokowanie możliwości zapisu danych może wywołać problemy związane z pracą aplikacji.

W zaprezentowanym wcześniej poleceniu `docker container run` sprawiliśmy, że wszystkie foldery poza wybranymi wolumenami były przeznaczone tylko do odczytu. Istnieje możliwość wykonania operacji odwrotnej, w której jedynie wybrany wolumen będzie przeznaczony tylko do odczytu (zastosowanie większej liczby argumentów `-v` pozwala na nadanie statusu tylko do odczytu większej liczbie kontenerów). Podczas korzystania z wolumenów warto pamiętać o tym, że przy montowaniu ich w kontenerze są one domyślnie montowane jako puste wolumeny i nadpisywany jest przy tym wybrany katalog kontenera. Mechanizm ten można obejść za pomocą argumentu `--volumes-from` lub poprzez dodanie danych do uruchomionego kontenera w inny sposób:

```
$ docker container run -d -v /folder/lokalny/html:/var/www/html:ro nginx
```

Powыższe polecenie montuje katalog */folder/lokalny/html* z systemu plików hosta Dockera w katalogu kontenera */var/www/html* i przypisuje mu atrybut tylko do odczytu. Rozwiązanie to przydaje się, gdy nie chcesz, aby aktywny kontener zapisywał dane w wolumenie, lub gdy chcesz, aby dane lub pliki konfiguracyjne nie były modyfikowane.

Polecenie diff

Przyjrzyjmy się jeszcze raz poleceniu `docker diff`. Obsługa tego polecenia jest związana z bezpieczeństwem kontenerów, a więc warto sprawdzić, czy działa ono poprawnie, za pomocą obrazów umieszczonych w serwisie Docker Hub lub innym pewnym repozytorium.

Pamiętaj o tym, że każda osoba mająca dostęp do hosta i demona Dockera ma również dostęp do wszystkich uruchomionych kontenerów Dockera. Jeżeli nie masz żadnego mechanizmu monitorującego host Dockera, to osoba mająca do niego dostęp może niepostrzeżenie uruchamiać polecenia w Twoich kontenerach i wykonywać złośliwe operacje.

Przyjrzyjmy się kontenerowi MySQL uruchomionemu w poprzedniej sekcji:

```
$ docker container diff mysql
```

Jak widzisz, polecenie to nie zwróciło żadnych plików. Dlaczego tak się dzieje? Polecenie `diff` informuje o zmianach wprowadzonych w kontenerze od momentu jego uruchomienia. W poprzedniej sekcji uruchamialiśmy kontener MySQL w trybie tylko do odczytu, a następnie montowaliśmy trzy wolumeny, które pozwalały na zapis danych podczas uruchamiania kontenera, a więc nie ma żadnych różnic pomiędzy systemami plików pobranego obrazu i uruchomionego kontenera.

Zatrzymaj i usuń kontener MySQL, uruchamiając następujące polecenia:

```
$ docker container stop mysql  
$ docker container rm mysql
```

Teraz uruchom ponownie ten sam kontener, ale nie korzystaj z flagi tylko do odczytu i nie definiuj wolumenów:

```
$ docker container run -d --name mysql -e MYSQL_ROOT_PASSWORD=password mysql  
$ docker container exec mysql touch /testowy_zapis_danych  
$ docker container diff mysql
```

Tym razem utworzono w kontenerze dwa foldery i dodano do niego kilka plików:

```
C /run/mysqld  
A /run/mysqld/mysqld.pid  
A /run/mysqld/mysqld.sock  
A /run/mysqld/mysqld.sock.lock  
C /tmp  
A /testowy_zapis_danych
```

Jak widzisz, zaprezentowana technika pozwala dostrzec podejrzane modyfikacje kontenera.

Dobre praktyki

W tym podrozdziale opiszymy dobre praktyki korzystania z Dockera, a także poruszymy zagadnienia związane z zabezpieczaniem dostępu do internetu zgodnie z zaleceniami Center for Internet Security.

Dobre praktyki pracy w Dockerze

Zanim przejdziemy do zaleceń organizacji Center for Internet Security, warto przyjrzeć się dobrym praktykom pracy w Dockerze:

- **Jedna aplikacja na kontener.** Dziel aplikacje tak, aby w żadnym z kontenerów nie uruchamiać więcej niż jednej aplikacji. Docker został stworzony z myślą o uruchamianiu pojedynczych aplikacji, a więc trzymanie się tej reguły pozwoli Ci uniknąć poważnych problemów. Izolacja aplikacji jest najważniejszą cechą Dockera.
- **Instaluj tylko to, czego potrzebujesz.** Zgodnie z tym, co pisaliśmy w poprzednich rozdziałach, w obrazach kontenerów instaluj tylko to, czego potrzebujesz. Jeżeli w celu umożliwienia pracy procesu uruchomionego w kontenerze musisz zainstalować dodatkowe oprogramowanie, to upewnij się, że jest to absolutnie konieczne. Minimalizacja instalowanych pakietów pozwala na zmniejszenie rozmiaru obrazów, zwiększenie ich przenośności, a także redukcję potencjalnych możliwości ataku.
- **Sprawdź, kto może uzyskać dostęp do hostów Dockera.** Każdy, kto posiada dostęp do hostów Dockera na poziomie administratora, może manipulować wszystkimi obrazami i kontenerami znajdującymi się w danym hoście.
- **Korzystaj z najnowszej wersji.** Zawsze korzystaj z najnowszej wersji Dockera. Dzięki temu masz pewność załatwania wszystkich luk bezpieczeństwa, a także dostępu do najnowszych funkcji. Korzystanie z najnowszej wersji Docker CE pozwala na zminimalizowanie zagrożeń związanych z bezpieczeństwem, ale wprowadzanie nowych funkcji może czasami być problematyczne. Jeżeli się tego obawiasz, to warto przyjrzeć się wersji LTS Enterprise dostępnej na stronie twórców Dockera i systemu Red Hat.
- **Korzystaj z dostępnych źródeł wiedzy.** Jeżeli potrzebujesz pomocy, możesz skorzystać z dużej wiedzy społeczności użytkowników Dockera. Planując własne środowisko Dockera i oceniąc aktywne platformy, korzystaj z różnych witryn internetowych, czytaj dokumentację, a także fora dyskusyjne Slack. Więcej informacji na temat forów dyskusyjnych i społeczności użytkowników Dockera znajdziesz w rozdziale 13. „Dalsze kroki z Dockerem”.

Zalecenia organizacji Center for Internet Security

Center for Internet Security (CSI) to niezależna organizacja non profit mająca na celu zwiększenie bezpieczeństwa korzystania z internetu. Organizacja ta publikuje wytyczne uznawane za najlepsze praktyki. Więcej informacji na jej temat znajdziesz na stronie <https://www.cisecurity.org/>.

Wytyczne organizacji CIS dotyczące dobrych praktyk korzystania z Dockera można pobrać za darmo ze strony <https://www.cisecurity.org/benchmark/docker/>. Obecnie jest to 196-stronicowy dokument na licencji *Creative Commons* dotyczący Dockera w wersji 1.13.0 i nowszych.

Do dokumentu tego zajrzesz po wykonaniu skanowania opisanego w dalszej części tego rozdziału, w celu określenia elementów, które należy poprawić. Poradnik jest podzielony na sekcje dotyczące następujących problemów:

- konfiguracja hosta;
- konfiguracja demona Dockera;
- pliki konfiguracyjne demona Dockera;
- obrazy i środowisko kontenerów;
- operacje bezpieczeństwa w Dockerze.

Konfiguracja hosta

Rozdział „Host configuration” dotyczy tematów związanych z konfiguracją hostów Dockera. Host to część środowiska Dockera, w której uruchomione są wszystkie kontenery. W związku z tym zabezpieczanie hosta jest jednym z najważniejszych zagadnień. Poprawnie skonfigurowany host tworzy pierwszą linię obrony przed atakami.

Konfiguracja demona Dockera

W rozdziale „Docker daemon configuration” znajdziesz zalecenia dotyczące bezpiecznego użytkowania demona Dockera. Każda zmiana konfiguracji demona Dockera wpływa na wszystkie kontenery. Znajdziesz tu opis opcji, o których pisaliśmy w tej książce wcześniej, a także elementy, które opiszemy w kolejnym podrozdziale przy okazji opisu narzędzia.

Pliki konfiguracyjne demona Dockera

Rozdział „Docker daemon configuration files” dotyczy tematyki związanej z plikami i katalogami używanymi przez demona Dockera. Znalazły się tu między innymi zagadnienia dotyczące uprawnień. Niektóre pliki tekstowe mogą zawierać poufne informacje, do których nie każdy powinien mieć dostęp.

Środowisko wykonawcze, obrazy i budowane pliki kontenerów

W rozdziale „Container images/runtime and build files” poruszone zagadnienia związane z załączaniem obrazów kontenerów, a także budowanych plików.

Na początku tego rozdziału poradnika opisano tematykę obrazów, obrazów bazowych i budowanych plików. Powtórzymy jeszcze raz: obrazy bazowe, a także obrazy rozbudowujące funkcjonalność infrastruktury muszą pochodzić z pewnych źródeł. Ponadto w tym rozdziale znajdujesz wytyczne dotyczące tworzenia własnych obrazów bazowych.

Środowisko robocze kontenerów

W przeszłości rozdział „Container runtime” wchodził w skład kolejnego rozdziału. Znajdziesz w nim wiele informacji dotyczących bezpieczeństwa.

Zachowaj ostrożność podczas korzystania ze zmiennych środowiskowych. Czasami mogą one zostać wykorzystane do przeprowadzenia ataku. Odsłanianie zbyt wielu elementów kontenera, takich jak szczegóły aplikacji, połączenia baz danych i zmienne środowiskowe, może negatywnie wpływać nie tylko na bezpieczeństwo kontenera, a także na bezpieczeństwo hosta i innych uruchomionych w nim kontenerów.

Operacje bezpieczeństwa w Dockerze

W rozdziale „Docker security operations” opisano zagadnienia dotyczące bezpieczeństwa wdrożeń. Zagadnienia te są związane z dobrymi praktykami pracy w Dockerze, a więc najlepiej, abyś stosował się do przedstawionych zaleceń.

Aplikacja Docker Bench Security

W tej sekcji opiszemy aplikację Docker Bench Security. Możesz zainstalować ją i uruchomić w swoim środowisku Dockera. Narzędzie to analizuje:

- konfigurację hosta;
- konfigurację demonika Dockera;
- pliki konfiguracyjne demonika Dockera;
- obrazy kontenerów;
- środowisko wykonawcze kontenera;
- operacje bezpieczeństwa platformy Docker.

Brzmi znajomo? Powinno, ponieważ niektóre z tych zagadnień były opisywane w sekcjach dotyczących zbioru dobrych praktyk. Aplikacja przeprowadza za Ciebie wiele skomplikowanych operacji i wyświetli ostrzeżenia i informacje dotyczące poszczególnych elementów konfiguracji środowiska Dockera (opisywane są nawet elementy, które poprawnie przejdą test).

Przyjrzymy się uruchamianiu narzędzia, przykładowi jego działania i przeanalizujemy zwrócone przez nie informacje.

Uruchamianie narzędzia w systemach macOS i Windows

Uruchamianie narzędzia jest proste. Jego autorzy udostępniają je w postaci kontenera Dockera. Ponadto autorzy udostępniają kod źródłowy, pozwalając na modyfikację zwracanych danych lub sposobu pracy narzędzia (wygenerowane dane mogą być np. wysłane na adres e-mail), ale na razie nie musimy przeprowadzać żadnych modyfikacji opisywanego narzędzia.

Narzędzie Docker Bench Security znajdziesz w repozytorium GitHub pod adresem <https://github.com/docker/docker-bench-security/>. W celu uruchomienia go w systemie macOS lub Windows wystarczy uruchomić następujące polecenie w Terminalu:

W poniższym poleceniu brakuje linii wymaganej do sprawdzenia funkcjonowania menedżera *systemd*, ponieważ system Moby Linux, na którym działa bezpośrednio Docker dla systemów macOS i Windows, nie korzysta z tego menedżera. Wkrótce przyjrzymy się pracy w systemie opartym na tym menedżerze.

```
docker run -it --net host --pid host --cap-add audit_control \
-e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
-v /var/lib:/var/lib \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /etc:/etc --label docker_bench_security \
docker/docker-bench-security
```

Po pobraniu obrazu zostanie on od razu uruchomiony. Przeprowadzona zostanie kontrola hosta Dockera, a jej wyniki pojawią się w oknie Terminala (patrz zrzut na następnej stronie).

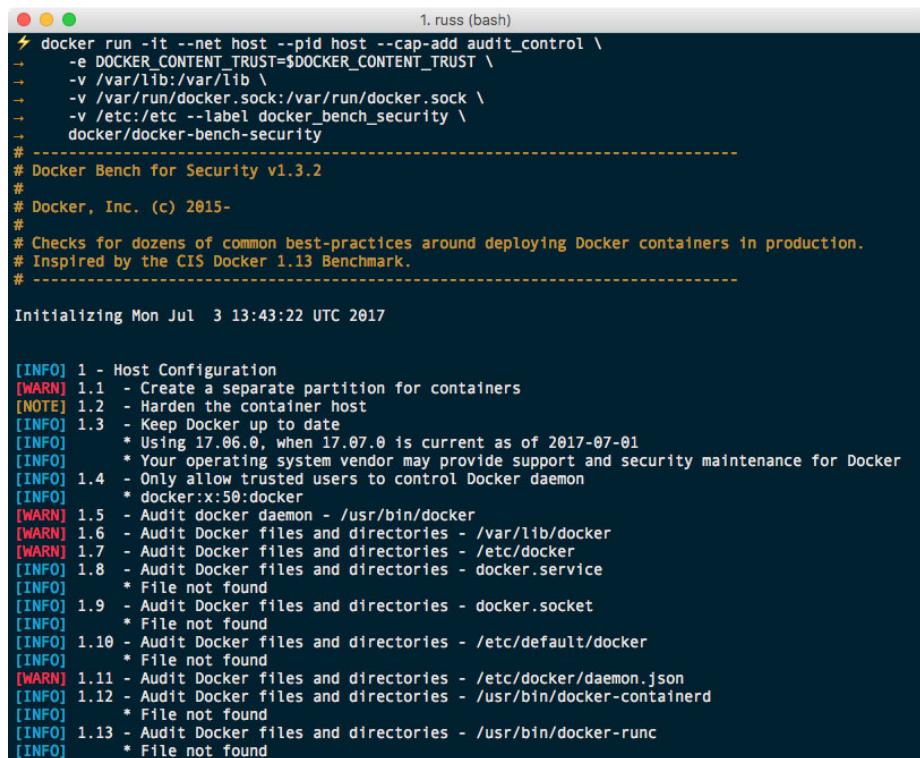
W wyświetlonych komunikatach znajduje się kilka ostrzeżeń (*[WARN]*), a także wskazówki (*[NOTE]*) i informacje (*[INFO]*). Host ten jest zarządzany przez Dockera, a więc zgodnie z oczekiwaniami nie ma tu wielu elementów, którymi musielibyśmy się przejmować.

Uruchamianie narzędzia w systemie Linux Ubuntu

Zanim przejdziemy do analizy wyników kontroli przeprowadzonej przez narzędzie Docker Bench Security, chciałbym uruchomić czysty serwer Ubuntu 17.04 i zainstalować Dockera od nowa. Po zainstalowaniu uruchomię kilka kontenerów, których konfiguracja nie będzie najlepsza.

Zdecydowałem się na uruchomienie dwóch następujących kontenerów z repozytorium Docker Hub:

```
$ docker container run -d --name root-nginx -v /:/mnt nginx
$ docker container run -d --name priv-nginx --privileged=true nginx
```



```
1. russ (bash)
$ docker run -it --net host --pid host --cap-add audit_control \
-e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
-v /var/lib:/var/lib \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /etc:/etc --label docker_bench_security \
docker/docker-bench-security
# -----
# Docker Bench for Security v1.3.2
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker 1.13 Benchmark.
# ----

Initializing Mon Jul  3 13:43:22 UTC 2017

[INFO] 1 - Host Configuration
[WARN] 1.1 - Create a separate partition for containers
[NOTE] 1.2 - Harden the container host
[INFO] 1.3 - Keep Docker up to date
[INFO] * Using 17.06.0, when 17.07.0 is current as of 2017-07-01
[INFO] * Your operating system vendor may provide support and security maintenance for Docker
[INFO] 1.4 - Only allow trusted users to control Docker daemon
[INFO] * docker:x:50:docker
[WARN] 1.5 - Audit docker daemon - /usr/bin/docker
[WARN] 1.6 - Audit Docker files and directories - /var/lib/docker
[WARN] 1.7 - Audit Docker files and directories - /etc/docker
[INFO] 1.8 - Audit Docker files and directories - docker.service
[INFO] * File not found
[INFO] 1.9 - Audit Docker files and directories - docker.socket
[INFO] * File not found
[INFO] 1.10 - Audit Docker files and directories - /etc/default/docker
[INFO] * File not found
[WARN] 1.11 - Audit Docker files and directories - /etc/docker/daemon.json
[INFO] 1.12 - Audit Docker files and directories - /usr/bin/docker-containerd
[INFO] * File not found
[INFO] 1.13 - Audit Docker files and directories - /usr/bin/docker-runc
[INFO] * File not found
```

Następnie zbudowałem własny obraz na bazie systemu Ubuntu w wersji 16.04, w którym uruchomiłem klienta SSH. Zrobiłem to za pomocą następującego polecenia:

```
$ docker container run -d -P --name sshd eg_sshd
```

Jak widzisz, w kontenerze *root-nginx* montuję główny katalog systemu plików hosta z pełnymi uprawnieniami do odczytu i zapisu danych. Ponadto kontener *priv-nginx* działa z rozszerzonymi uprawnieniami, a kontener *sshd* korzysta z protokołu SSH.

W celu uruchomienia procedury sprawdzania hosta Dockera Ubuntu korzystam z polecenia:

```
docker run -it --net host --pid host --cap-add audit_control \
-e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
-v /var/lib:/var/lib \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /usr/lib/systemd:/usr/lib/systemd \
-v /etc:/etc --label docker_bench_security \
docker/docker-bench-security
```

Korzystamy z systemu operacyjnego obsługującego menedżera *systemd*, a więc w celu przeprowadzenia jego kontroli montujemy folder */usr/lib/systemd*.

Narzędzie Docker Bench Security po przeprowadzeniu kontroli zwraca wiele informacji. Co one oznaczają? Przyjrzymy się kolejnym sekcjom zwróconych danych.

Analiza zwracanych informacji

Narzędzie Docker Bench Security zwraca następujące typy informacji:

- **[PASS]** (poprawne) — tu znajdziesz elementy, które są poprawne. Nie musisz zwracać na nie większej uwagi, ale warto je przejrzeć, aby poczuć satysfakcję wynikającą z poprawnie wykonanego zadania. Im więcej takich elementów, tym lepiej.
- **[WARN]** (ostrzeżenia) — tu znajdziesz elementy, które należy poprawić. Komunikaty tego typu są niepożądane.
- **[INFO]** (informacje) — tu znajdziesz elementy, którym warto się przyjrzeć i które warto poprawić (jeżeli dojdzieś do wniosku, że są one ważne z punktu widzenia Twojej konfiguracji i potrzeb bezpieczeństwa).
- **[NOTE]** (wskazówka) — tu znajdziesz wskazówki dotyczące dobrych praktyk.

Wyniki przeprowadzonego audytu są dzielone na sześć sekcji zawierających informacje o:

- konfiguracji hosta;
- konfiguracji demona Dockera;
- plikach konfiguracyjnych demona Dockera;
- obrazach kontenerów;
- środowisku roboczym kontenerów;
- operacjach bezpieczeństwa Dockera.

Przyjrzyjmy się informacjom umieszczonym we wszystkich sekcjach. Opisywane przeze mnie wyniki powstały w wyniku skanowania domyślnego hosta Dockera Ubuntu bez żadnych zmian konfiguracji systemu. Skupimy się na ostrzeżeniach wyświetlanych w każdej sekcji. Jeżeli przeprowadzisz audit we własnym środowisku, to możesz trafić na inne ostrzeżenia niż te, które zostaną opisane w kolejnych sekcjach. Chcemy skupić się na ostrzeżeniach wyświetlanych na początku u praktycznie każdego użytkownika.

Konfiguracja hosta

W sekcji *Host configuration* znaleźliśmy pięć ostrzeżeń:

[WARN] 1.1 - Create a separate partition for containers

Domyślnie Docker umieszcza wszystkie swoje pliki w folderze `/var/lib/docker` znajdującym się w systemie plików hosta (znajdują się tam wszystkie obrazy, kontenery i wolumeny utworzone za pomocą domyślnego sterownika). W związku z tym objętość tego folderu może szybko się zwiększać. Nasz host działa na pojedynczej partycji i wykonywanie operacji na kontenerach może doprowadzić do zapełnienia całego dysku, co może wywołać niestabilność hosta.

[WARN] 1.5 - Audit docker daemon - /usr/bin/docker

[WARN] 1.6 - Audit Docker files and directories - /var/lib/docker

[WARN] 1.7 - Audit Docker files and directories - /etc/docker

[WARN] 1.10 - Audit Docker files and directories - /etc/default/docker

Ostrzeżenia te wyróżnione flagami, ponieważ nie zainstalowano oprogramowania *auditd* i nie ma reguł audytu demona Dockera i związanych z nim plików.Więcej informacji na temat oprogramowania *auditd* znajdziesz na stronie <https://www.linux.com/learn/customized-file-monitoring-auditd>.

Konfiguracja demona Dockera

W mojej sekcji *Docker daemon configuration* aż siedem elementów oznaczono flagą ostrzeżenia ([WARN]):

[WARN] 2.1 - Restrict network traffic between containers

Domyślnie Docker pozwala na nieograniczony przepływ ruchu sieciowego pomiędzy kontenerami tego samego hosta. Oczywiście można to zmienić.Więcej informacji na temat obsługi sieci znajdziesz na stronie <https://docs.docker.com/network/>.

[WARN] 2.8 - Enable user namespace support

Domyślnie przestrzeń nazw użytkownika nie jest mapowana. Mapowanie tej przestrzeni jest możliwe, ale może obecnie doprowadzić do problemów związanych z kilkoma funkcjami Dockera.Więcej informacji na temat znanych ograniczeń znajdziesz na stronie <https://docs.docker.com/engine/reference/commandline/dockerd/#user-namespace-known-restrictions>.

[WARN] 2.11 - Use authorization plugin

W przypadku domyślnej instalacji Dockera dostęp do demona Dockera jest nieograniczony. Dostęp ten można ograniczyć do uwierzytelnionych użytkowników, włączając wtyczkę autoryzacji.Więcej informacji na ten temat znajdziesz na stronie https://docs.docker.com/engine/extend/plugins_authorization/.

[WARN] 2.12 - Configure centralized and remote logging

Korzystam tylko z jednego hosta i nie posługuję się usługą typu *rsyslog* w celu dostarczenia danych dziennika Dockera do centralnego serwera. Ponadto nie skonfigurowałem sterownika dziennika demona Dockera.Więcej informacji na ten temat znajdziesz na stronie <https://docs.docker.com/config/containers/logging/configure/>.

[WARN] 2.13 - Disable operations on legacy registry (v1)

W momencie pisania tej książki domyślna instalacja Dockera jest skonfigurowana tak, aby mogła komunikować się z przestarzałym interfejsem *V1 Docker Registry API*, a także nowszym interfejsem *V2 Docker Registry API*.Więcej informacji na temat wyłączenia obsługi starego interfejsu znajdziesz na stronie <https://docs.docker.com/engine/reference/commandline/dockerd/#legacy-registries>.

[WARN] 2.14 - Enable live restore

Flaga *--live-restore* aktywuje pełną obsługę kontenerów pracujących bez demona. Kontenery te nie zostają zatrzymane w momencie wyłączenia demona. Pracują one dalej, a demon

po ponownym uruchomieniu może się z nimi połączyć. Domyślnie opcja ta nie jest włączona z powodu problemu z kompatybilnością. Więcej informacji na ten temat znajdziesz na stronie <https://docs.docker.com/config/containers/live-restore/>.

[WARN] 2.18 - Disable Userland Proxy

Kontenery mogą komunikować się ze światem zewnętrznym za pomocą techniki *hairpin NAT* lub pośredniczącego serwera użytkownika. W przypadku większości instalacji preferowana jest pierwsza z tych opcji, ponieważ korzysta z programu sterującego *iptables* i charakteryzuje się lepszą wydajnością. Większość instalacji Dockera zrealizowanych w nowoczesnych systemach operacyjnych obsługuje tę technikę. Więcej informacji na temat wyłączenia obsługi komunikacji przy użyciu pośredniczącego serwera użytkownika znajdziesz na stronie <https://docs.docker.com/config/containers/container-networking/>.

Pliki konfiguracyjne demona Dockera

W sekcji *Docker daemon configuration files* nie znaleźliśmy żadnych ostrzeżeń.

Obrazy kontenerów

W sekcji *Container images and build files* znalazły się trzy ostrzeżenia. Komunikaty zajmujące więcej niż jedną linię są poprzedzane znakiem *:

[WARN] 4.1 - Create a user for the container
 [WARN] * Running as root: sshd
 [WARN] * Running as root: priv-nginx
 [WARN] * Running as root: root-nginx

Procesy w moich kontenerach działają z uprawnieniami administratora (jest to domyślne zachowanie większości kontenerów). Więcej informacji na ten temat znajdziesz na stronie <https://docs.docker.com/engine/security/security/>.

[WARN] 4.5 - Enable Content trust for Docker

Włączenie obsługi zaufanych treści zapewnia sprawdzanie cyfrowych podpisów pobieranych kontenerów, dzięki czemu zawsze uruchamiasz właściwe obrazy. Więcej informacji na ten temat znajdziesz na stronie https://docs.docker.com/engine/security/trust/content_trust/.

[WARN] 4.6 - Add HEALTHCHECK instruction to the container image
 [WARN] * No Healthcheck found: [eg_sshd:latest]
 [WARN] * No Healthcheck found: [nginx:latest]
 [WARN] * No Healthcheck found: [ubuntu:16.04]

Podczas budowy własnych obrazów możesz utworzyć procedurę sprawdzania ich statusu, co pozwoli na okresowe sprawdzanie statusu kontenera po uruchomieniu go na podstawie zawartości obrazu (dzięki temu kontener może zostać w razie konieczności uruchomiony ponownie). Więcej informacji na ten temat znajdziesz na stronie <https://docs.docker.com/engine/reference/builder/#healthcheck>.

Środowisko robocze kontenerów

Uruchamiając kontenery, celowo popechnięlem pewne błędy, a więc spodziewam się wystąpienia wielu ostrzeżeń. W sekcji *Container runtime* znalazłem aż 11 ostrzeżeń.

```
[WARN] 5.2 - Verify SELinux security options, if applicable  
[WARN] * No SecurityOptions Found: sshd  
[WARN] * No SecurityOptions Found: root-nginx
```

Poniżej zaprezentowano wynik fałszywie pozytywny (w punkcie 5.1 umieszczone komunikat [PASS]). Korzystam z systemu Ubuntu, a SELinux jest stosowany tylko w przypadku komputerów pracujących pod kontrolą systemu Red Hat:

```
[PASS] 5.1 - Do not disable AppArmor Profile
```

Dwa kolejne ostrzeżenia są wynikiem przeprowadzonych przeze mnie operacji:

```
[WARN] 5.4 - Do not use privileged containers  
[WARN] * Container running in Privileged mode: priv-nginx
```

Dotyczy to również poniższych ostrzeżeń:

```
[WARN] 5.6 - Do not run ssh within containers  
[WARN] * Container running sshd: sshd
```

Można je bezpiecznie zignorować. Bardzo rzadko zachodzi konieczność uruchamiania kontenera działającego w trybie uprzywilejowanym. Korzystanie z tego trybu jest niezbędne tylko wtedy, gdy kontener musi nawiązać interakcję z silnikiem Docker Engine uruchomionym w hoście — dochodzi do tego np. wtedy, gdy korzystasz z graficznego interfejsu takiego jak Portainer (zagadnienie to opisaliśmy w rozdziale 8. „Portainer”).

Pisaliśmy wcześniej o tym, że w kontenerach nie powinno uruchamiać się obsługą protokołu SSH. Co prawda może być to konieczne w niektórych sieciach, ale są to sytuacje wyjątkowe.

Kolejne dwa statusy oznaczono flagą ostrzeżenia (*[WARN]*), ponieważ domyślnie wszystkie kontenery uruchomione w hoście dzielą zasoby w równych ilościach. Zdefiniowanie ograniczeń w dostępie do pamięci i czasu pracy procesora sprawi, że kontenerom o wyższym priorytecie nie będzie brakowało zasobów (zasoby te nie zostaną zagarnięte przez kontenery o niższym priorytecie):

```
[WARN] 5.10 - Limit memory usage for container  
[WARN] * Container running without memory restrictions: sshd  
[WARN] * Container running without memory restrictions: priv-nginx  
[WARN] * Container running without memory restrictions: root-nginx  
  
[WARN] 5.11 - Set container CPU priority appropriately  
[WARN] * Container running without CPU restrictions: sshd  
[WARN] * Container running without CPU restrictions: priv-nginx  
[WARN] * Container running without CPU restrictions: root-nginx
```

Przypominam, że o ile to tylko możliwe, należy uruchamiać kontenery w trybie tylko do odczytu, a wszelkie dane, które muszą być zapisane, należy kierować do zamontowanych wolumenów:

```
[WARN] 5.12 - Mount container's root filesystem as read only
[WARN] * Container running with root FS mounted R/W: sshd
[WARN] * Container running with root FS mounted R/W: priv-nginx
[WARN] * Container running with root FS mounted R/W: root-nginx
```

Poniższe ostrzeżenia są spowodowane tym, że odsłonięty port nie został powiązany z określonym adresem IP hosta Dockera:

```
[WARN] 5.13 - Bind incoming container traffic to a specific host interface
[WARN] * Port being bound to wildcard IP: 0.0.0.0 in sshd
```

W moim hoście uruchomiłem tylko jeden interfejs, ale gdyby host ten miał wiele interfejsów, to kontener mógłby zostać udostępniony w wielu sieciach, co mogłoby być problemem, gdybym korzystał z sieci zewnętrznej i wewnętrznej. Więcej informacji na ten temat znajdziesz na stronie <https://docs.docker.com/network/>.

```
[WARN] 5.14 - Set the 'on-failure' container restart policy to 5
[WARN] * MaximumRetryCount is not set to 5: sshd
[WARN] * MaximumRetryCount is not set to 5: priv-nginx
[WARN] * MaximumRetryCount is not set to 5: root-nginx
```

Podczas uruchamiania kontenera nie użyłem flagi `--restart`, ale parametr `MaximumRetryCount` nie ma domyślnej wartości. W związku z tym, gdyby kontener miał ciągle problemy, to Docker próbowałby go w kółko uruchomić ponownie. Oczywiście miałoby to negatywny wpływ na hosta Dockera. Przypisanie parametrowi `MaximumRetryCount` wartości 5 sprawiłoby, że próba ponownego uruchomienia kontenera byłaby podejmowana maksymalnie pięciokrotnie.

```
[WARN] 5.25 - Restrict container from acquiring additional privileges
[WARN] * Privileges not restricted: sshd
[WARN] * Privileges not restricted: priv-nginx
[WARN] * Privileges not restricted: root-nginx
```

Docker domyślnie nie ogranicza swojego procesu ani procesów pochodnych uzyskujących nowe przywileje za pośrednictwem danych binarnych `suid` i `sgid`. Informacje na temat sposobów zatrzymania tego mechanizmu znajdziesz na stronie <http://www.projectatomic.io/blog/2016/03/no-new-privs-docker/>.

```
[WARN] 5.26 - Check container health at runtime
[WARN] * Health check not set: sshd
[WARN] * Health check not set: priv-nginx
[WARN] * Health check not set: root-nginx
```

Znow nie aktywowaliśmy mechanizmu sprawdzającego okresowo status kontenerów. Na forum repozytorium GitHub pod adresem <https://github.com/moby/moby/pull/22719> znajdziesz opis rozwiązania umożliwiającego wygenerowanie takiego mechanizmu.

```
[WARN] 5.28 - Use PIDs cgroup limit
[WARN] * PIDs limit not set: sshd
```

```
[WARN] * PIDs limit not set: priv-nginx  
[WARN] * PIDs limit not set: root-nginx
```

Istnieje możliwość przeprowadzenia ataku typu fork-bomba za pomocą pojedynczego polecenia uruchomionego wewnątrz kontenera. Może to doprowadzić do zawieszenia się hosta Dockera. W takim przypadku jedynym ratunkiem jest ponowne uruchomienie hosta. Aby zapobiec się przed takim atakiem, należy skorzystać z flagi `--pids-limit`. Więcej informacji na ten temat znajdziesz w wątku: <https://github.com/moby/moby/pull/18697/>.

Operacje bezpieczeństwa Dockera

W sekcji *Docker security operations* znajdują się tylko informacje o dobrych praktykach:

```
[INFO] 6.1 - Perform regular security audits of your host system and containers  
[INFO] 6.2 - Monitor Docker containers usage, performance and metering  
[INFO] 6.3 - Backup container data  
[INFO] 6.4 - Avoid image sprawl  
[INFO] * There are currently: 4 images  
[INFO] 6.5 - Avoid container sprawl  
[INFO] * There are currently a total of 8 containers, with 4 of them currently running
```

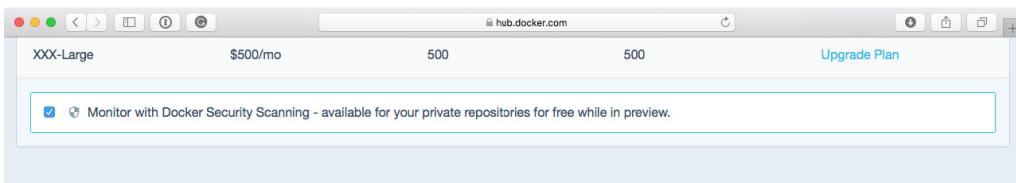
Docker Bench — podsumowanie

Jak widzisz, sprawdzenie hosta Dockera narzędziem Docker Bench pozwala ustalić, czy host spełnia zalecenia CIS Docker Benchmark. Bez tego narzędzia musielibyśmy ręcznie przeprowadzać każdy test opisany w dokumencie składającym się ze 196 stron.

Skanowanie zabezpieczeń Dockera

Uslugi Docker Cloud i Docker Hub mają wbudowane funkcje skanowania bezpieczeństwa prywatnych obrazów. W chwili pisania tej książki funkcja ta jest dostępna testowo za darmo dla użytkowników posiadających prywatne repozytoria, ale czas darmowego korzystania z niej jest ograniczony.

W celu rozpoczęcia korzystania z tej usługi wejdź na stronę *Plan* swojego konta Docker Hub i zaznacz poniższe pole wyboru:



Po zaznaczeniu tej opcji załadowanie obrazu do prywatnego repozytorium spowoduje przeskanowanie go. Wyniki przeprowadzonego testu znajdziesz w sekcji *Repositories* (repozytoria) chmury Docker Cloud. Jak widzisz, darmowa wersja tego mechanizmu nie jest jeszcze w pełni dopracowana.

Oto przykład wyników skanowania:

Gdy w sprawdzanym obrazie znalezione zostaną słabe punkty, możesz go kliknąć w celu wyświetlenia bardziej szczegółowych informacji na temat wykrytych problemów:

| Component | Vulnerability | Severity |
|---|--|----------------------------------|
| pcre 8.40-r2 BSD: Permissive License | CVE-2017-7245 CVE-2017-7246 CVE-2017-7186 CVE-2017-7244 | Major Major Major Major |
| bzip2 1.0.6 BSD: Permissive License | CVE-2016-3189 | Major |

Ze zrzutu wynika, że mój kontener NGINX musi zostać zaktualizowany, ponieważ wiadomo, że zainstalowane wersje pakietów *pcre* i *bzip2* mogą być problematyczne.

Raport zawiera również numery problemów — **Common Vulnerabilities and Exposures (CVE)** — i odwołania pozwalające na znalezienie szczegółowych informacji w bazie **National Vulnerability Database (NVD)**, którą można znaleźć na stronie <http://cve.mitre.org/>. Dzięki temu możliwe jest dokładne przeanalizowanie ryzyka i jego potencjalnego wpływu na działanie aplikacji.

Darmowy testowy dostęp do funkcji skanującej jest możliwy od około roku i dziwi mnie, że funkcja ta jest nadal ukryta, mimo że pozwala na przeprowadzenie dokładnej statycznej analizy obrazów. Mam nadzieję, że wkrótce twórcy Dockera utworzą na jej podstawie samodzielna usługę.

Statyczna analiza kodu jest metodologią poprawiania kodu i plików binarnych bez uruchamiania analizowanego kodu. Kod nie jest uruchamiany, a więc technika ta doskonale nadaje się do sprawdzania zagadnień związanych z bezpieczeństwem, ponieważ nie wymaga uruchamiania problematycznych programów i narażania hosta na niebezpieczeństwo w trakcie analizy. Więcej informacji na ten temat znajdziesz na stronie https://www.owasp.org/index.php/Static_Code_Analysis.

Niezależne usługi poprawiające bezpieczeństwo

Na koniec tego rozdziału przyjrzymy się wybranym niezależnym usługom pomagającym w ocenie bezpieczeństwa obrazów.

Quay

W rozdziale 3. „Przechowywanie obrazów i ich dystrybucja” opisywaliśmy Quay — repozytorium obrazów firmy CoreOS, ale nie pisaliśmy o tym, że repozytorium to przeprowadza operację skanowania obrazów po ich załadowaniu lub zbudowaniu.

Wyniki operacji skanowania znajdziesz w sekcji *Repository Tags* (etykiety nadane przez repozytorium) opisującej wybrany obraz. W sekcji tej umieszczono kolumnę *Security Scan* (skanowanie bezpieczeństwa). Z poniższego zrzutu wynika, że w utworzonym przeze mnie przykładowym obrazie nie znaleziono żadnych problemów:

| TAG | LAST MODIFIED | SECURITY SCAN | SIZE | IMAGE |
|--------|---------------|---------------|--------|---------------------|
| latest | 3 days ago | Passed | 2.4 MB | SHA256 3be68a02ab74 |
| master | 3 days ago | Passed | 2.4 MB | SHA256 f2321d8a34c5 |

Kliknięcie przycisków znajdujących się w miejscach, w których obecnie umieszczono napis *Passed* (zaliczone), spowoduje wyświetlenie szczegółowych informacji na temat każdego z wykrytych zagrożeń. W aktualnym obrazie nie wykryto żadnych zagrożeń (to dobrze), a więc na ekranie wyświetlono mało danych. Kliknięcie ikony paczki (*Package*) znajdującej się w menu wyświetlanym po lewej stronie ekranu spowoduje przejście do listy pakietów wykrytych

podczas skanowania. W moim testowym obrazie znaleziono 13 pakietów i w żadnym z nich nie wykryto problemów. Obok każdego pakietu wyświetlana jest informacja o jego wersji i sposobie wprowadzenia go do obrazu.

The screenshot shows the Quay Security Scanner results for the repository `russmckendrick/dockerfile-example`. The main message is "Quay Security Scanner has recognized **13** packages." and "13 packages with no vulnerabilities." A large green circle icon indicates a 100% scan completion. Below this, there's a table titled "Image Packages" listing 13 packages with their details:

| PACKAGE NAME | PACKAGE VERSION | VULNERABILITIES | REMAINING AFTER UPGRADE | UPGRADE IMPACT | INTRODUCED IN IMAGE |
|--------------|-----------------|-----------------|-------------------------|----------------|---|
| apk-tools | 2.7.2-r0 | None Detected | (N/A) | (N/A) | ADD file:4583e12bf5coec40b861a3... |
| musl-utils | 1.1.16-r10 | None Detected | (N/A) | (N/A) | ADD file:4583e12bf5coec40b861a3... |
| musl | 1.1.16-r10 | None Detected | (N/A) | (N/A) | ADD file:4583e12bf5coec40b861a3... |
| alpine-keys | 2.1-r1 | None Detected | (N/A) | (N/A) | ADD file:4583e12bf5coec40b861a3... |
| zlib | 1.2.11-r0 | None Detected | (N/A) | (N/A) | ADD file:4583e12bf5coec40b861a3... |
| nginx | 1.12.0-r2 | None Detected | (N/A) | (N/A) | RUN apk add --update nginx & r... |
| busybox | 1.26.2-r5 | None Detected | (N/A) | (N/A) | ADD |

At the bottom right of the table, there's a "Contact Us" button.

Jak widzisz, narzędzie Quay skanuje publicznie udostępniane obrazy w darmowej wersji reprezytorium. Skanowanie bezpieczeństwa jest standardowym elementem wszystkich planów serwisu Quay. Skanowanie to, podobnie jak zaprezentowana wcześniej usługa skanowania obrazów Dockera, jest analizą statyczną.

Więcej informacji na temat Quay znajdziesz na stronie <https://quay.io/>.

Clair

Clair to otwarty projekt twórców systemu CoreOs. Ogólnie rzecz biorąc, jest to usługa prowadząca statyczną analizę obrazów umieszczonych w darmowych i komercyjnych reprezytoriach Quay.

Usługa ta działa, tworząc lokalną kopię następujących baz danych zbierających informacje o zagrożeniach:

- **Debian Security Bug Tracker:** <https://security-tracker.debian.org/tracker/>;
- **Ubuntu CVE Tracker:** <https://launchpad.net/ubuntu-cve-tracker/>;

- **Red Hat Security Data:** <https://www.redhat.com/security/data/metrics/>;
- **Oracle Linux Security Data:** <https://linux.oracle.com/security/>;
- **Alpine SecDB:** <https://git.alpinelinux.org/cgit/alpine-secdb/>;
- **NIST NVD:** <https://nvd.nist.gov/>.

Po utworzeniu kopii danych montowany jest system plików obrazu, a następnie przeprowadzane jest skanowanie zainstalowanych pakietów i porównywanie ich z sygnaturami pochodząymi z powyższych źródeł.

Clair nie jest usługą prostą w obsłudze. Jest ona wyposażona w interfejs programistyczny. Nie ma standardowych graficznych interfejsów sieciowych lub narzędzi umożliwiających korzystanie z niej z poziomu wiersza poleceń. Dokumentację interfejsu programistycznego Clair znajdziesz na stronie https://coreos.com/clair/docs/latest/api_v1.html.

Instrukcję instalacji znajdziesz w serwisie GitHub pod adresem <https://github.com/coreos/clair>. Listę narzędzi obsługujących Clair umieszczono na stronie <https://coreos.com/clair/docs/latest/integrations.html>.

Podsumowanie

W rozdziale tym poruszyliśmy wybrane zagadnienia związane z bezpieczeństwem Dockera. Najpierw wymieniliśmy rzeczy, które należy wziąć pod uwagę podczas uruchamiania kontenerów (porównaliśmy uruchamianie kontenerów z uruchamianiem maszyn wirtualnych). Przyjrzyliśmy się zaletom hosta Dockera i wyjaśniliśmy, dlaczego warto korzystać tylko z obrazów pochodzących z pewnego źródła. Ponadto opisaliśmy polecenia, które przydają się podczas zabezpieczania środowiska Dockera.

W celu zminimalizowania potencjalnych szkód wywołanych przez intruza uruchomiliśmy kontener w trybie tylko do odczytu. Nie wszystkie aplikacje można uruchomić w tym trybie, a więc opisaliśmy proces śledzenia zmian zachodzących w kontenerze od chwili jego uruchomienia. Mechanizm ten pozwala na wykrycie wielu problemów.

W dalszej części rozdziału opisaliśmy wytyczne bezpieczeństwa Dockera opracowane przez Center for Internet Security. Wytyczne te dotyczą wielu aspektów środowiska Dockera. Na koniec przyjrzyliśmy się narzędziu Docker Bench for Security. Opisaliśmy proces jego uruchamiania, a także przedstawiliśmy analizę przykładowych danych zwracanych przez to narzędzie. Dane te zostały podzielone na sześć sekcji zawierających informacje dotyczące konfiguracji hosta, konfiguracji demona Dockera, plików konfiguracyjnych demona Dockera, obrazów kontenerów, środowiska roboczego kontenera i operacji bezpieczeństwa Dockera.

W kolejnym rozdziale przyjrzymy się przepływowi zadań w Dockerze i technikom pracy z kontenerami.

Przepływ zadań w platformie Docker

W tym rozdziale przyjrzymy się różnym przepływom zadań w platformie Docker. Podczas lektury tego rozdziału połączysz wszystkie elementy i zaczniesz swobodnie korzystać z Dockera we własnym środowisku produkcyjnym. W tym rozdziale opiszemy następujące zagadnienia:

- Docker i prace programistyczne;
- monitorowanie platformy Docker;
- rozszerzanie za pomocą zewnętrznych platform;
- środowisko produkcyjne.

Docker i prace programistyczne

Analiżę przepływów zadań rozpoczęmy od przyjrzenia się temu, jak Docker może pomóc programistom. Już na samym początku rozdziału 1. „Docker — wprowadzenie”, w sekcji „Czym jest Docker?”, opisaliśmy problem spotykany podczas pracy programistów. Nie opisaliśmy jeszcze w pełni jego rozwiązania, a więc zróbmy to teraz.

Przyjrzymy się temu, jak programiści mogą pracować nad projektem WordPress na swoim lokalnym komputerze za pomocą Dockera uruchomionego w systemie macOS lub Windows oraz narzędzia Docker Compose.

Chcemy uzyskać działające środowisko WordPress. W tym celu wykonany następujące operacje:

1. Pobierzemy i zainstalujemy środowisko WordPress.

2. Umożliwimy dostęp do plików WordPressa edytorom takim jak Atom (<https://atom.io/>), Visual Studio Code (<https://code.visualstudio.com/>) lub Sublime Text (<https://www.sublimetext.com/>) uruchomionym w środowisku lokalnym.
3. Skonfigurujemy WordPressa za pomocą narzędzia WP-CLI (<http://wp-cli.org/>).
4. Umożliwimy zatrzymywanie, uruchamianie, a nawet usuwanie kontenerów bez utraty efektów pracy.

Przed zainstalowaniem WordPressa warto przyjrzeć się plikowi Docker Compose i uruchamianym aplikacjom:

Poniższy kod znajduje się w folderze `r12\docker-wordpress`. Folder ten wchodzi w skład archiwum, które możesz pobrać ze strony <ftp://ftp.helion.pl/przyklady/dockaz.zip>.

```
version: "3"

services:

  web:
    image: nginx:alpine
    ports:
      - "8080:80"
    volumes:
      - "./wordpress/web:/var/www/html"
      - "./wordpress/nginx.conf:/etc/nginx/conf.d/default.conf"
    depends_on:
      - wordpress

  wordpress:
    image: wordpress:php7.1-fpm-alpine
    volumes:
      - "./wordpress/web:/var/www/html"
    depends_on:
      - mysql

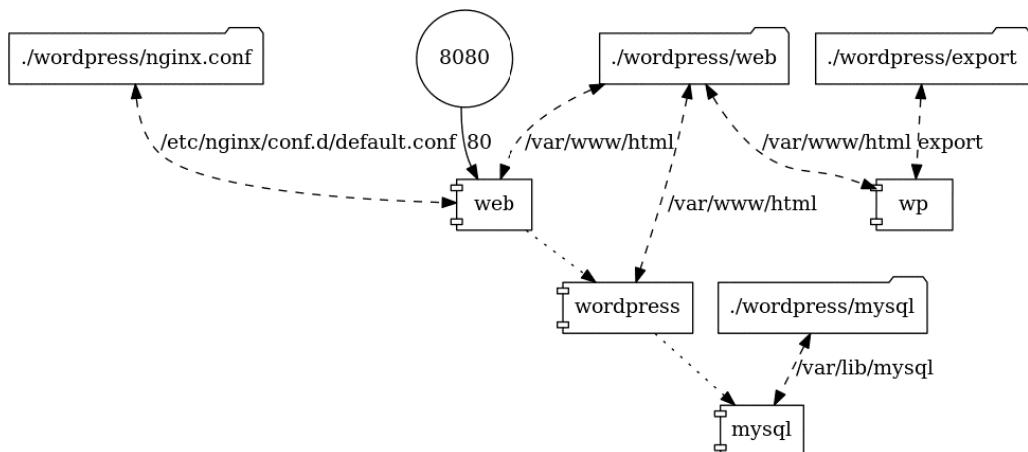
  mysql:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: "wordpress"
      MYSQL_USER: "wordpress"
      MYSQL_PASSWORD: "wordpress"
      MYSQL_DATABASE: "wordpress"
    volumes:
      - "./wordpress/mysql:/var/lib/mysql"

  wp:
    image: wordpress:cli-php7.1
    volumes:
      - "./wordpress/web:/var/www/html"
      - "./wordpress/export:/export"
```

Zawartość tego pliku Docker Compose można przedstawić w formie graficznej za pomocą narzędzia *docker-compose-viz* (<https://github.com/pmsipilot/docker-compose-viz>) opracowanego przez PMSIpilot. W celu wykonania takiej wizualizacji uruchom poniższe polecenie w tym samym folderze, w którym znajduje się plik *docker-compose.yml*:

```
$ docker run --rm -it --name dcv -v $(pwd):/input pmsipilot/docker-compose-viz
  ↳render -m image docker-compose.yml
```

Wygenerowany zostanie plik o nazwie *docker-compose.png*, w którym znajdzie się następująca wizualizacja:



Narzędzie *docker-compose-viz* może być użyte w celu wygenerowania wizualizacji dowolnego pliku Docker Compose. Z zaprezentowanej grafiki wynika, że w naszym pliku zdefiniowano cztery usługi.

Pierwsza z nich (usługa o nazwie *web*) jest jedyną, która może korzystać z sieci hosta. Tworzy ona element graniczny instalacji WordPressa. Usługa ta jest oparta na oficjalnym obrazie serwera NGINX pobranym ze strony <https://store.docker.com/images/nginx/>. Ma ona dwie funkcje.

Przyjrzyjmy się konfiguracji serwera NGINX:

```

server {
    server_name _;
    listen 80 default_server;

    root /var/www/html;
    index index.php index.html;

    access_log /dev/stdout;
    error_log /dev/stdout info;

    location / {
        try_files $uri $uri/ /index.php?$args;
    }
}
```

```
}

location ~ .php$ {
    include fastcgi_params;
    fastcgi_pass wordpress:9000;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_buffers 16 16k;
    fastcgi_buffer_size 32k;
}
}
```

Jak widzisz, cała treść (poza PHP) jest pobierana z folderu `/var/www/html/`, który jest zamontowany z systemu plików hosta, a wszystkie żądania plików PHP są obsługiwane przez naszą drugą usługę. Usłudze tej nadano nazwę `wordpress`. Działa ona na porcie nr 9000. Konfiguracja samego serwera NGINX jest montowana z maszyny hosta do pliku `/etc/nginx/conf.d/default.conf`.

Druga usługa o nazwie `wordpress` jest oparta na oficjalnym obrazie WordPressa pobranym ze strony <https://store.docker.com/images/wordpress>. Skorzystałem z etykiety `php7.1-fpm-alpine`, a więc WordPress działa na PHP 7.1 i korzysta z menedżera PHP-FPM opartego na bazie systemu Alpine Linux.

Menedżer **PHP-FPM (FastCGI Process Manager)** to implementacja interfejsu FastCGI napisana w PHP, która oferuje wiele doskonałych funkcji. W naszym przypadku umożliwia ona uruchomienie PHP jako usługi, której możemy przypisać port, a następnie używać go do przekazywania żądań. Rozwiążanie to wpisuje się w zasadę uruchamiania jednej usługi w każdym z kontenerów. Więcej informacji na temat menedżera PHP-FPM znajdziesz na stronie <https://php-fpm.org/>.

Montujemy główny katalog usługi sieciowej — w systemie plików hosta jest to katalog `wordpress/web`, a w kontenerze usługi jest to katalog `/var/www/html/`. Na początku folder znajdujący się w systemie plików hosta będzie pusty, ale po uruchomieniu usługi `wordpress` wykryty zostanie brak instalacji rdzenia WordPress, który zostanie skopiowany do tego właśnie folderu, co umożliwi uruchomienie instalacji WordPressa i skopiowanie go do hosta w celu przygotowania go do pracy.

Kolejna usługa, `mysql`, korzysta z oficjalnego obrazu MySQL (<https://store.docker.com/images/mysql>). Jest to jedyny z czterech używanych w tym przykładzie obrazów, który nie jest oparty na systemie Alpine Linux (drodzy autorzy MySQL-a, zlitujcie się w końcu i udostępnijcie obraz oparty na systemie Alpine Linux!). Przekazujemy kilka zmiennych środowiskowych, dzięki którym podczas pierwszego uruchomienia kontenera zostanie utworzona baza danych, a także nazwa użytkownika i hasło. Jeżeli chcesz korzystać z tej bazy we własnych projektach, to pamiętaj o zmianie hasła.

Ponownie montujemy folder z systemu plików hosta. W tym przypadku folder `wordpress/mysql` jest montowany w ścieżce `/var/lib/mysql` będącej domyślnym folderem, w którym MySQL przechowuje bazy danych i związane z nimi pliki.

Po uruchomieniu kontenera w folderze *wordpress/mysql* pojawi się kilka plików. Nie polecam edytowania ich za pomocą lokalnego środowiska programistycznego.

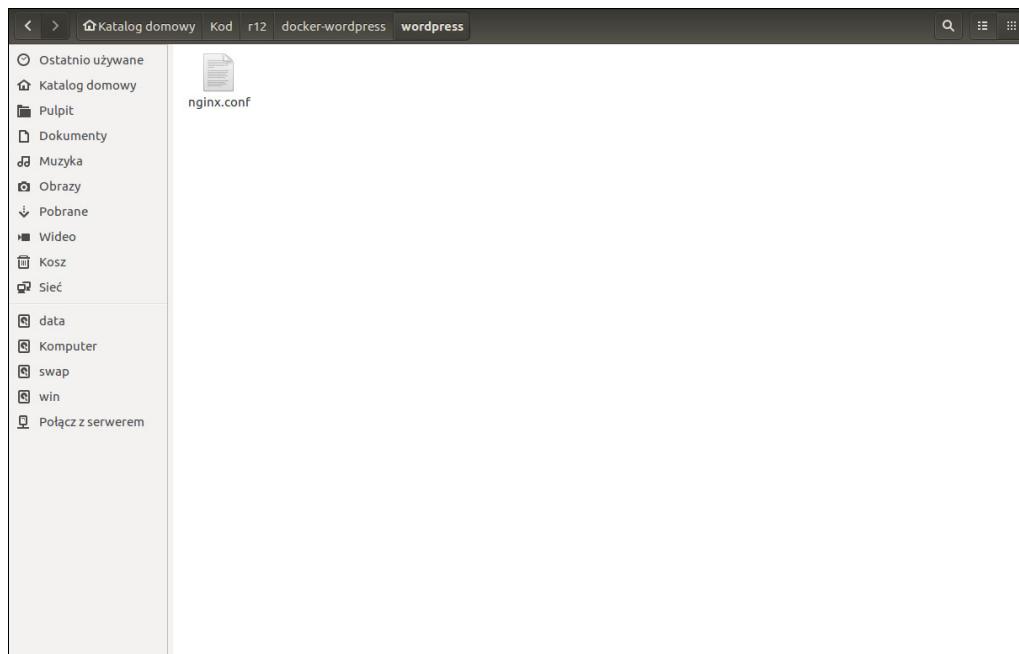
Ostatniej usługę nadalem nazwę *wp*. Różni się ona od pozostałych trzech usług tym, że zostanie zakończona od razu po uruchomieniu. Wynika to z tego, że w jej kontenerze nie ma żadnego procesu wykonywanego przez dłuższy czas. Usługa ta zapewnia dostęp do narzędzia wiersza poleceń WordPressa w środowisku, które jest dokładnie dopasowane do naszego głównego kontenera *wordpress*.

Podobnie jak w przypadku kontenerów *web* i *wordpress*, montujemy główny katalog witryny, a także drugi katalog o nazwie */export* — zagadnienie to opiszemy bardziej szczegółowo po skonfigurowaniu systemu WordPress.

W celu uruchomienia systemu WordPress skorzystaj z następujących poleceń:

```
$ docker-compose pull  
$ docker-compose up -d  
$ docker-compose ps
```

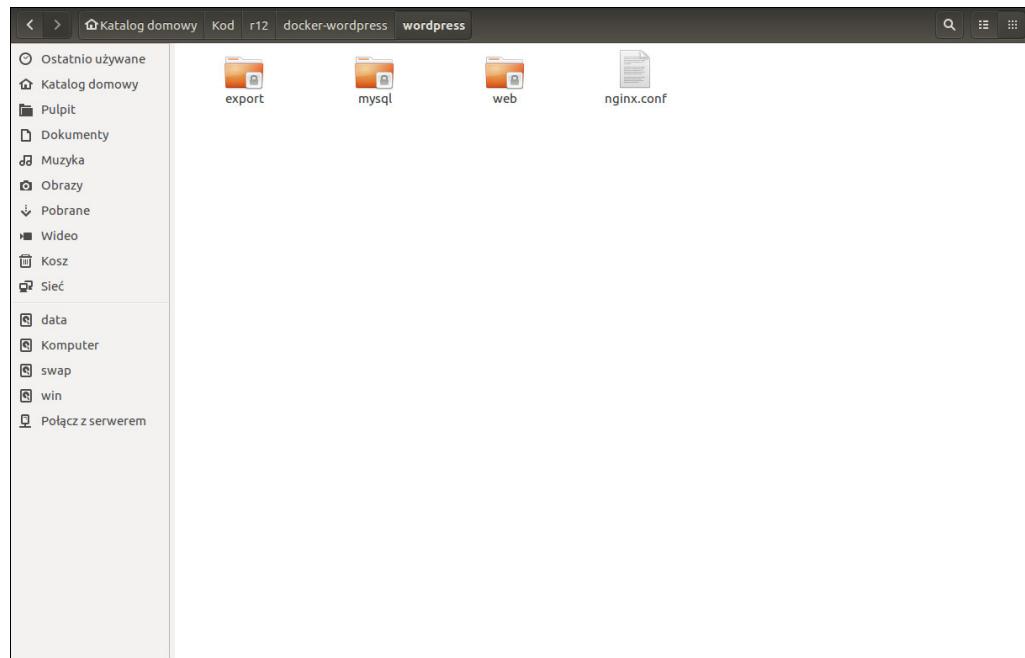
Polecenia te spowodują pobranie obrazów i uruchomienie usług *web*, *wordpress* i *mysql*, a także przygotowanie usługi *wp*. Przed uruchomieniem usług zawartość folderu *wordpress* wygląda następująco:



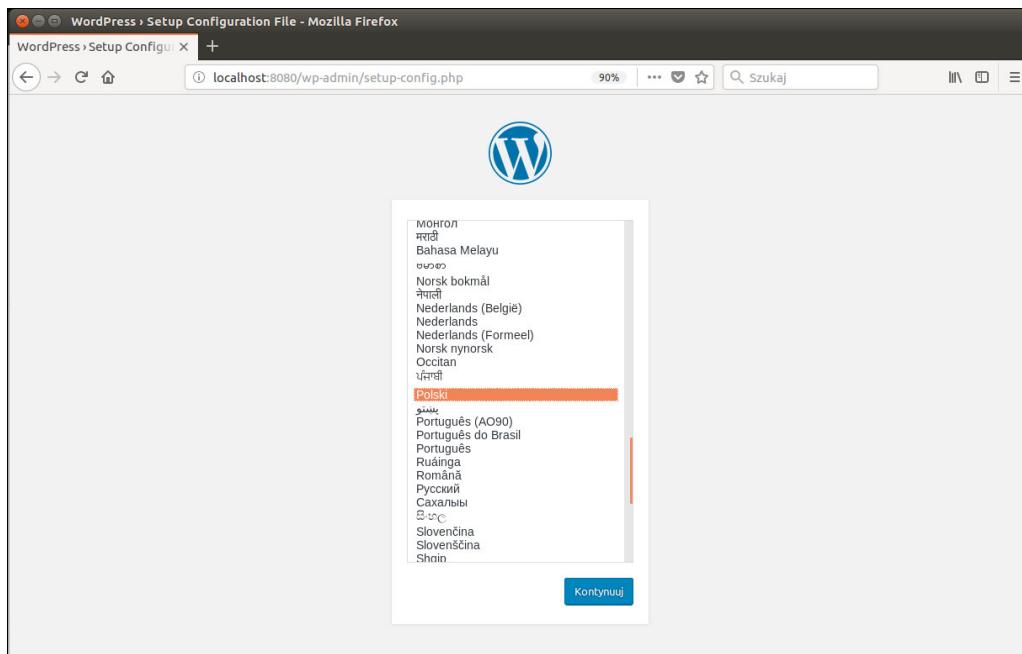
Jak widzisz, w folderze znajduje się tylko plik *nginx.conf*. Oto komunikaty wyświetlane w oknie Terminala podczas uruchamiania usługi:

```
1. docker-wordpress (bash)
⚡ docker-compose up -d
Creating network "dockerwordpress_default" with the default driver
Creating dockerwordpress_wp_1 ...
Creating dockerwordpress_mysql_1 ...
Creating dockerwordpress_wp_1
Creating dockerwordpress_mysql_1 ... done
Creating dockerwordpress_wordpress_1 ...
Creating dockerwordpress_wordpress_1 ... done
Creating dockerwordpress_web_1 ...
Creating dockerwordpress_web_1 ... done
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ docker-compose ps
          Name           Command    State     Ports
----- 
dockerwordpress_mysql_1   docker-entrypoint.sh mysqld   Up      3306/tcp
dockerwordpress_web_1     nginx -g daemon off;   Up      0.0.0.0:8080->80/tcp
dockerwordpress_wordpress_1 docker-entrypoint.sh php-fpm Up      9000/tcp
dockerwordpress_wp_1      docker-entrypoint.sh wp shell Exit 1
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ ⚡
```

Teraz w folderze *wordpress* powinny zostać utworzone trzy podkatalogi: *export*, *mysql* i *web*. Ponadto kontener *dockerwordpress_wp_1* powinien być zatrzymany (status *exit*).



Otwórz przeglądarkę i wejdź na stronę `localhost:8080/`. Powinien się pojawić ekran powitalny instalatora systemu WordPress, który umożliwia wybór używanego podczas instalacji:



Nie klikaj przycisku *Kontynuuj*, ponieważ przeniesie Cię to do kolejnego ekranu graficznego interfejsu instalatora. Zamiast tego wróć do okna Terminala.

Zamiast kończyć instalację za pomocą graficznego instalatora, skorzystamy z narzędzia *WP-CLI*. Musimy wykonać dwie operacje. Pierwsza z nich to tworzenie pliku *wp-config.php*. W celu utworzenia tego pliku uruchom następujące polecenie:

```
$ docker-compose run wp core config --dbname=wordpress --dbuser=wordpress --
  --dbpass=wordpress --dbhost=mysql --dbprefix=wp_
```

Przed uruchomieniem tej instrukcji dysponowałeś tylko plikiem *wp-config-sample.php* dołączonym do WordPressa, a teraz utworzyłeś własny plik konfiguracyjny *wp-config.php*:

```
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ ls -lhat wordpress/web/ | grep wp-config
-rw-r--r--@ 1 russ staff 2.8K 16 Dec 2015 wp-config-sample.php
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ docker-compose run wp core config --dbname=wordpress --dbuser=wordpress --dbpass=wordpress --dbhost=mysql --dbprefix=wp_
Success: Generated 'wp-config.php' file.
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ ls -lhat wordpress/web/ | grep wp-config
-rw-r--r-- 1 russ staff 2.5K 8 Jul 13:20 wp-config.php
-rw-r--r--@ 1 russ staff 2.8K 16 Dec 2015 wp-config-sample.php
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
```

W ostatnim poleceniu przekazaliśmy parametry bazy danych zdefiniowane w pliku Docker Compose i informujemy platformę WordPress o tym, że może połączyć się z usługą bazy danych znajdującej się pod adresem *mysql*.

Po skonfigurowaniu połączenia z bazą danych musimy skonfigurować naszą stronę WordPressa, a także utworzyć konto administratora i jego hasło. W tym celu uruchomimy następujące polecenie:

```
$ docker-compose run wp core install --url="http://localhost:8080" --title="Mój blog"  
↳--admin_user="admin" --admin_password="password" --admin_email="email@domain.com"
```

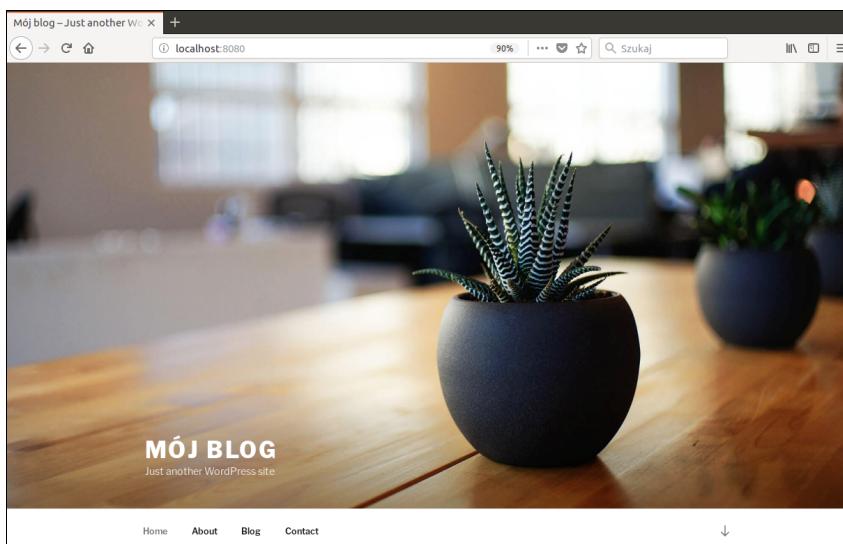
Po uruchomieniu tego polecenia wyświetlony zostanie komunikat informujący o braku usługi obsługującej pocztę elektroniczną. Nie przejmuj się. Na razie nie potrzebujemy funkcji wysyłania e-maili z WordPressa:

```
1. docker-wordpress (bash)  
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*  
⚡ docker-compose run wp core install --url="http://localhost:8080" --title="Mój blog" --admin_use  
r="admin" --admin_password="password" --admin_email="email@domain.com"  
sendmail: can't connect to remote host (127.0.0.1): Connection refused  
Success: WordPress installed successfully.  
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*  
⚡
```

Narzędzie WP-CLI zostało użyte w celu skonfigurowania następnych parametrów WordPressa:

- adres strony: *http://localhost:8080*;
- tytuł strony: *Mój blog*;
- nazwa użytkownika będącego administratorem: *admin*;
- hasło administratora: *password*;
- adres e-mail administratora: *email@domain.com*.

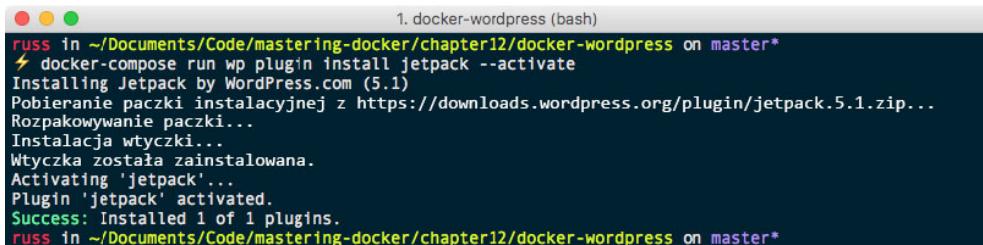
Teraz gdy otworzysz adres *http://localhost:8080*, w przeglądarce powinna pojawić się pusta strona WordPressa:



Zanim zaczniemy robić coś ze stroną, aktywujmy wtyczkę JetPack (<https://en-gb.wordpress.org/plugins/jetpack/>):

```
$ docker-compose run wp plugin install jetpack --activate
```

Uruchomienie tego polecenia spowoduje wyświetlenie następujących komunikatów:

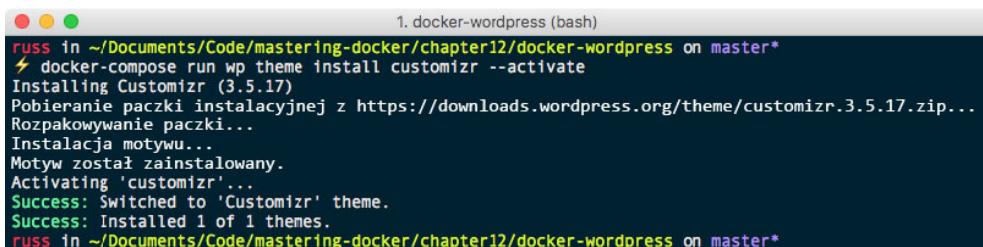


```
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ docker-compose run wp plugin install jetpack --activate
Installing Jetpack by WordPress.com (5.1)
Pobieranie paczki instalacyjnej z https://downloads.wordpress.org/plugin/jetpack.5.1.zip...
Rozpakowywanie paczki...
Instalacja wtyczki...
Wtyczka została zainstalowana.
Activating 'jetpack'...
Plugin 'jetpack' activated.
Success: Installed 1 of 1 plugins.
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
```

Teraz możemy zainstalować i włączyć motyw *Customizr* (<https://en-gb.wordpress.org/themes/customizr/>):

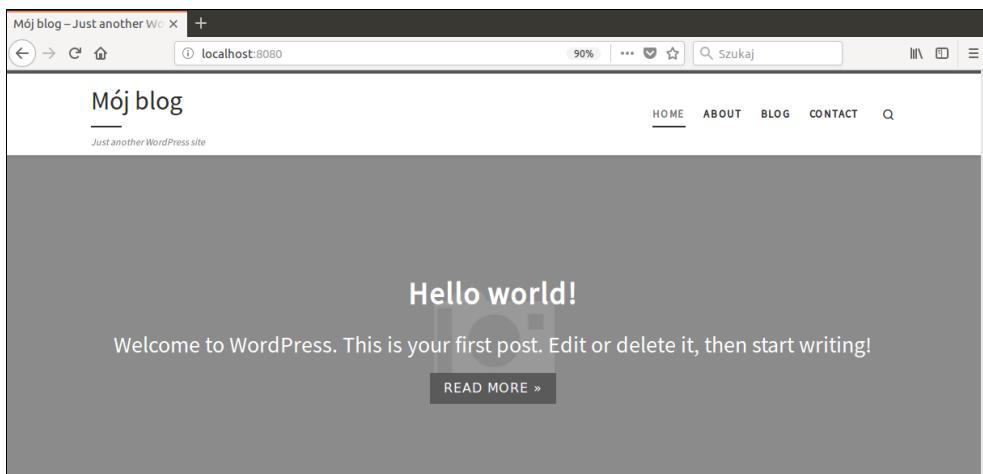
```
$ docker-compose run wp theme install customizr --activate
```

Po uruchomieniu tego polecenia na ekranie zobaczysz następujące komunikaty:



```
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ docker-compose run wp theme install customizr --activate
Installing Customizr (3.5.17)
Pobieranie paczki instalacyjnej z https://downloads.wordpress.org/theme/customizr.3.5.17.zip...
Rozpakowywanie paczki...
Instalacja motywu...
Motyw został zainstalowany.
Activating 'customizr'...
Success: Switched to 'Customizr' theme.
Success: Installed 1 of 1 themes.
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
```

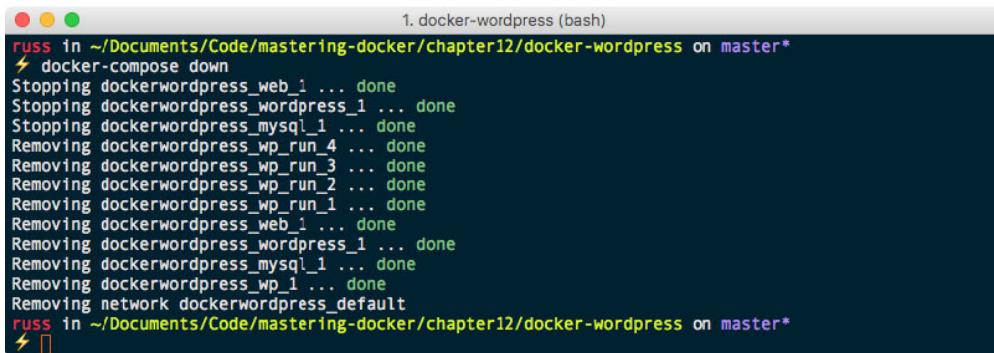
Odśwież w przeglądarce stronę <http://localhost:8080>. Teraz jej zawartość wygląda inaczej:



Zanim otworzymy edytor kodu, zatrzymajmy kontenery, w których działa WordPress, za pomocą następującego polecenia:

```
$ docker-compose down
```

Po jego uruchomieniu na ekranie pojawią się komunikaty:



```
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ docker-compose down
Stopping dockerwordpress_web_1 ... done
Stopping dockerwordpress_wordpress_1 ... done
Stopping dockerwordpress_mysql_1 ... done
Removing dockerwordpress_wp_run_4 ... done
Removing dockerwordpress_wp_run_3 ... done
Removing dockerwordpress_wp_run_2 ... done
Removing dockerwordpress_wp_run_1 ... done
Removing dockerwordpress_web_1 ... done
Removing dockerwordpress_wordpress_1 ... done
Removing dockerwordpress_mysql_1 ... done
Removing dockerwordpress_wp_1 ... done
Removing network dockerwordpress_default
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ ⚡
```

Cała instalacja WordPressa, łącznie z plikami i bazą danych, jest przechowywana w lokalnym systemie plików, a więc możemy przywrócić stan strony sprzed zatrzymania kontenerów za pomocą polecenia:

```
$ docker-compose up -d
```

Wejdźmy na stronę <http://localhost:8080>, aby sprawdzić, czy wszystko działa poprawnie. Otwórz folder *docker-wordpress* w wybranym przez siebie edytorze kodu. Ja korzystam z programu Microsoft Visual Studio Code.

W edytorze otwórz plik *wordpress/web/wp-blog-header.php* i do pierwszej instrukcji PHP dodaj poniższy kod, a następnie zapisz zmiany:

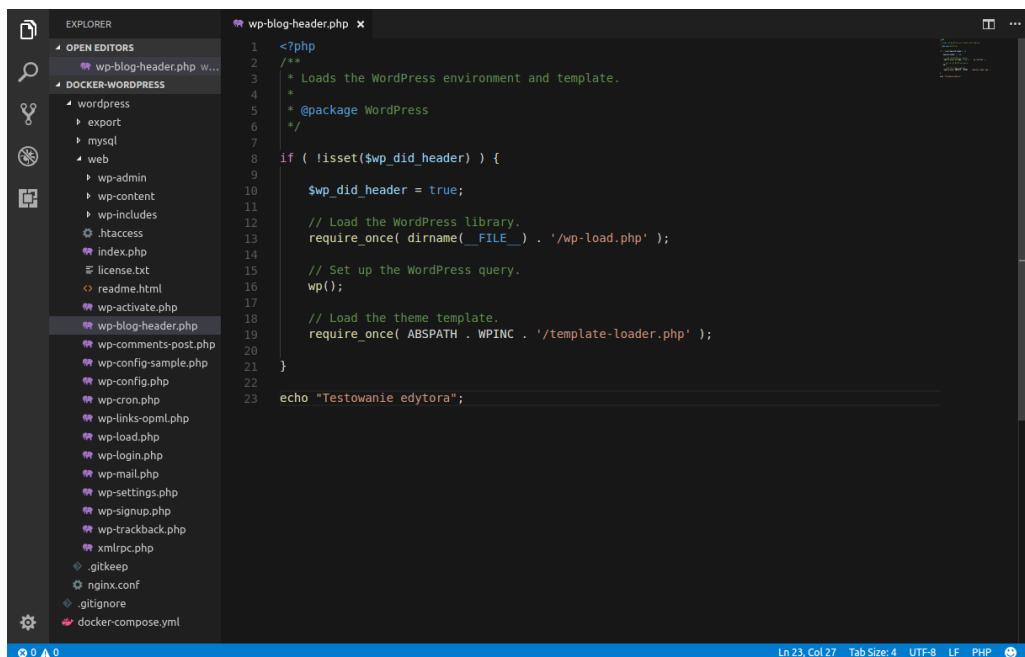
```
echo "Testowanie edytora";
```

Tak wygląda zmodyfikowany plik (patrz pierwszy zrzut na następnej stronie).

Po zapisaniu wprowadzonych zmian odśwież stronę wyświetlającą w przeglądarce. Napis *Testowanie edytora* powinien pojawić się na samym dole strony. Oto zrzut prezentujący powiększoną dolną część strony (napis jest mały i można mieć problemy z jego odczytaniem bez powiększenia zawartości strony) (patrz drugi zrzut na następnej stronie).

Na koniec zajmiemy się przyczyną montowania folderu *wordpress/export* w kontenerze *wp*.

Pamiętaj o tym, że nie warto edytować zawartości folderu *wordpress/mysql*. Nie powinno się go również udostępniać. Co prawda spakowanie zawartości tego folderu i wysłanie go współpracownikowi nie byłoby problematyczne, ale nie jest to uważaane za dobrą praktykę. Jest to powód montowania folderu *export*. Po zamontowaniu tego folderu narzędzie WP-CLI może wykonać zrzut zawartości bazy danych i zimportować go.

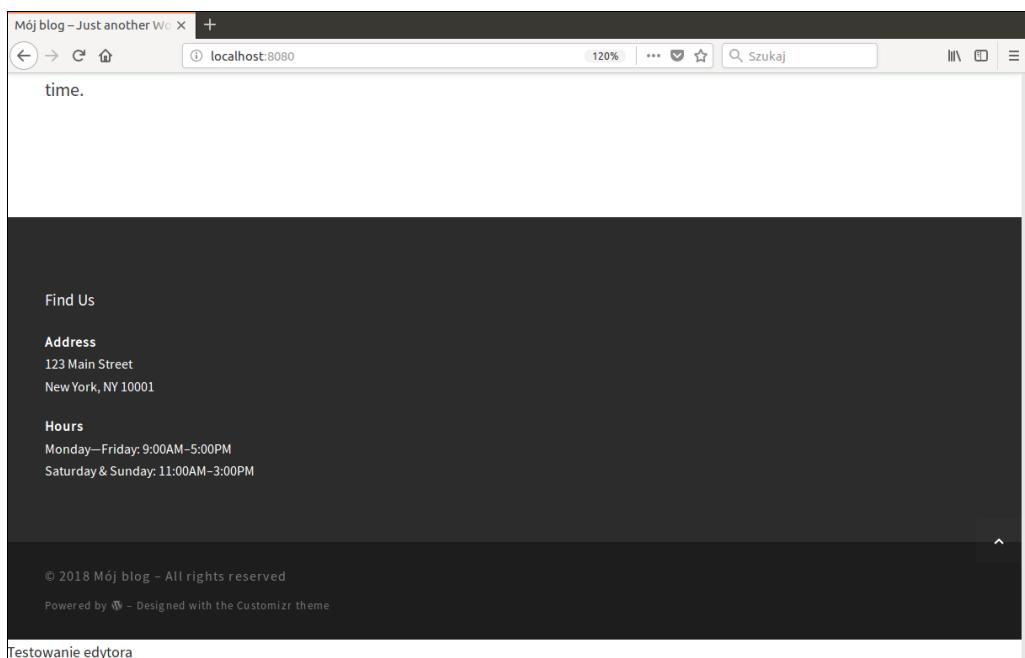


```

wp-blog-header.php x
1 <?php
2 /**
3  * Loads the WordPress environment and template.
4  *
5  * @package WordPress
6 */
7
8 if ( !isset($wp_did_header) ) {
9
10     $wp_did_header = true;
11
12     // Load the WordPress library.
13     require_once( dirname(__FILE__) . '/wp-load.php' );
14
15     // Set up the WordPress query.
16     wp();
17
18     // Load the theme template.
19     require_once(ABSPATH . WPINC . '/template-loader.php' );
20 }
21
22
23 echo "Testowanie edytora";

```

Ln 23, Col 27 Tab Size: 4 UTF-8 LF PHP ☀



Mój blog - Just another WordPress site

localhost:8080

time.

Find Us

Address
123 Main Street
New York, NY 10001

Hours
Monday–Friday: 9:00AM–5:00PM
Saturday & Sunday: 11:00AM–3:00PM

© 2018 Mój blog – All rights reserved

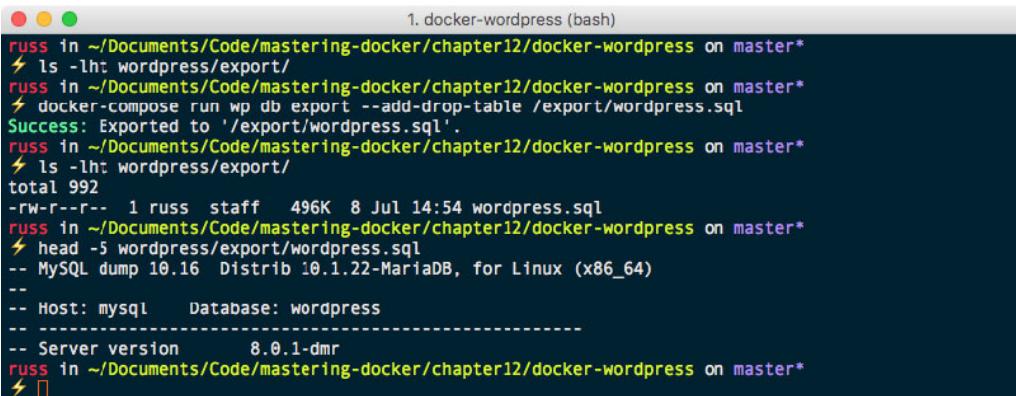
Powered by  – Designed with the Customizr theme

Testowanie edytora

W celu wykonania tej operacji uruchom polecenie:

```
$ docker-compose run wp db export --add-drop-table /export/wordpress.sql
```

Oto okno Terminala przedstawiające proces eksportu, a także zawartość folderu *wordpress/export* przed i po eksportie oraz kilka linii zrzutu bazy MySQL:

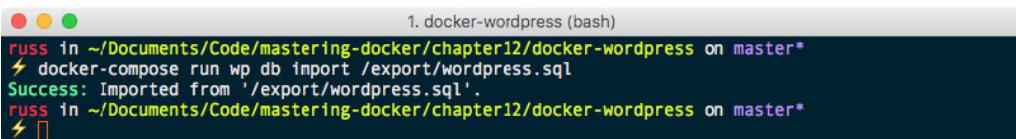


```
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ ls -lht wordpress/export/
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ docker-compose run wp db export --add-drop-table /export/wordpress.sql
Success: Exported to '/export/wordpress.sql'.
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ ls -lht wordpress/export/
total 992
-rw-r--r-- 1 russ staff 496K 8 Jul 14:54 wordpress.sql
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ head -5 wordpress/export/wordpress.sql
-- MySQL dump 10.16 Distrib 10.1.22-MariaDB, for Linux (x86_64)
--
-- Host: mysql      Database: wordpress
-----
-- Server version     8.0.1-dmr
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡
```

Gdybym w przyszłości popełnił błąd i chciał przywrócić zapisaną przed chwilą wersję bazy danych, to mógłbym to zrobić za pomocą polecenia:

```
$ docker-compose run wp db import /export/wordpress.sql
```

Uruchomienie go spowoduje wyświetlenie następujących komunikatów:



```
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡ docker-compose run wp db import /export/wordpress.sql
Success: Imported from '/export/wordpress.sql'.
russ in ~/Documents/Code/mastering-docker/chapter12/docker-wordpress on master*
⚡
```

Prześledziliśmy proces instalacji WordPressa, konfigurowania go za pomocą narzędzia WP-CLI, a także przeglądarki. Ponadto pokazaliśmy możliwość edytowania kodu, a także tworzenia kopii zapasowej bazy danych i przywracania jej. Wszystkie te operacje zostały wykonane bez potrzeby tracenia czasu na instalację i konfigurację narzędzi NGINX, PHP, MySQL i WP-CLI. Ponadto było się bez logowania do kontenera. Montowanie wolumenów z systemu plików hosta sprawiło, że efekty pracy nad stroną nie są tracone po zatrzymaniu kontenerów WordPressa.

Ponadto projekt zarządzany w taki właśnie sposób można za pomocą jednego polecenia skopiować i przekazać współpracownikowi, który ma zainstalowanego Dockera. Kod zostanie uruchomiony na innym komputerze w dokładnie takim samym środowisku.

Korzystaliśmy z oficjalnych obrazów pochodzących z repozytorium Docker Store, a więc możemy je bezpiecznie wdrożyć w środowisku produkcyjnym, ponieważ zostały one zbudowane z zachowaniem dobrych praktyk użytkowania Dockera.

Pamiętaj o konieczności zatrzymania i usunięcia kontenerów WordPressa za pomocą polecenia docker-compose down.

Monitorowanie

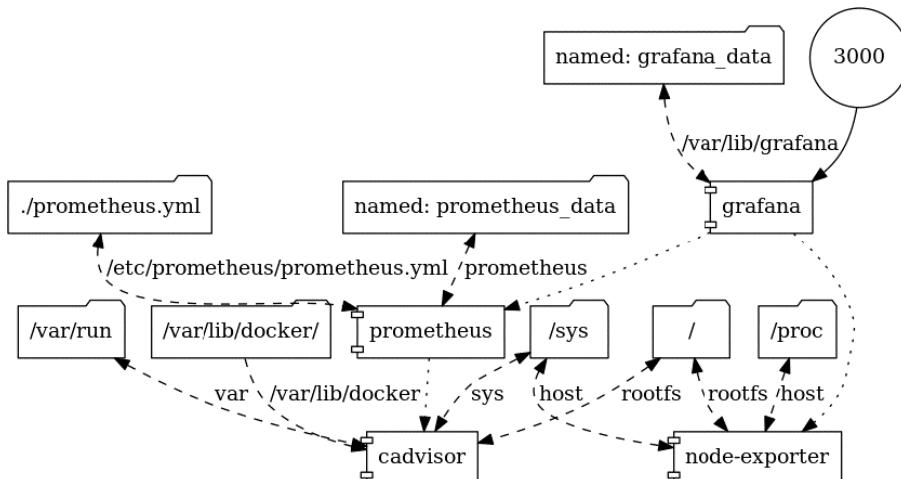
Teraz przyjrzymy się zagadnieniom związanym z monitorowaniem kontenerów i hostów Dockera. W rozdziale 4. „Zarządzanie kontenerami” opisaliśmy polecenia docker container top i docker container stats. Przypominamy, że polecenia te pokazują informacje zbierane w czasie rzeczywistym, nie przechowują żadnych danych historycznych.

Polecenia te przydają się podczas rozwiązywania problemów podczas pracy kontenerów i oceny poprawności ich pracy, ale nie pozwalają na dokładniejszą analizę problemów za pomocą danych zebranych z szerszego zakresu czasu pracy kontenera. Kontenery można skonfigurować tak, aby w razie zawieszenia się były automatycznie uruchamiane ponownie, ale rozwiązanie to uniemożliwia określenie przyczyn problemów związanych z ich pracą.

Będziemy korzystać z dość skomplikowanej konfiguracji. Istnieją alternatywne rozwiązania, a także trwają prace nad uproszczeniem tej konfiguracji w kolejnych wersjach silnika Docker Engine, ale więcej informacji na ten temat przedstawimy nieco później.

W archiwum pobranym ze strony <ftp://ftp.helion.pl/przykłady/dockaz.zip>, w katalogu r12, znajduje się folder o nazwie *prometheus*, w którym umieściłem plik Docker Compose uruchamiający cztery różne kontenery usług niezbędnych do monitorowania kontenerów, a także hostów Dockera.

Zamiast zaglądać bezpośrednio do pliku Docker Compose, przyjrzyjmy się jego wizualizacji:



Jak widzisz, plik Docker Compose jest dość skomplikowany — uruchamia cztery usługi:

- *cadvisor*: <https://github.com/google/cadvisor/>;
- *node-exporter*: https://github.com/prometheus/node_exporter/;
- *prometheus*: <https://prometheus.io/>;
- *grafana*: <https://grafana.com/>.

Przed uruchomieniem i skonfigurowaniem usług wyjaśnimy, dlaczego każda z nich jest potrzebna. Zaczniemy od usługi *cadvisor*.

Cadvisor to projekt firmy Google. Sekcja tej usługi pliku Docker File wygląda następująco:

```
  cAdvisor:  
    image: google/cadvisor:latest  
    container_name: cAdvisor  
    volumes:  
      - /:/rootfs:ro  
      - /var/run:/var/run:rw  
      - /sys:/sys:ro  
      - /var/lib/docker/:/var/lib/docker:ro  
    restart: unless-stopped  
    expose:  
      - 8080
```

Jak widzisz, dochodzi do montowania różnych części systemu plików hosta. Zabieg ten pozwoli narzędziu *cadvisor* uzyskać dostęp do środowiska Dockera w sposób podobny do tego, który zaprezentowaliśmy w rozdziale 8. „Portainer”. Dostęp ten jest niezbędny, ponieważ będziemy korzystać z narzędzia *cadvisor* w celu zbierania statystyk pracy kontenerów. Kontener *cadvisor* może pełnić rolę samodzielnej usługi monitorującej kontenery, ale nie chcemy go udostępniać w sieci publicznej. Udostępnimy go tylko innym kontenerom wchodząącym w skład naszego stosu.

Narzędzie *cadvisor* jest samodzielnią nakładką korzystającą z polecenia `docker container stat`, która wyświetla w formie graficznej informacje o kontenerach i jest wyposażona w prosty w obsłudze interfejs. Niestety przedstawia ona informacje o pracy kontenera w okresie nie dłuższym niż 5 minut. Chcemy gromadzić parametry pracy kontenerów w przeciągu kilku godzin, a nawet dni. W związku z tym musimy skorzystać z dodatkowych narzędzi zapisujących te parametry.

Narzędzie *cadvisor* wyświetli informacje, które chcemy gromadzić po przejściu pod adres <http://cAdvisor:8080/metrics/>. Za chwilę wyjaśnimy, dlaczego parametry te są ważne.

Kolejną uruchamianą przez nas usługą jest *node-exporter*. Oto jej definicja umieszczona w pliku Docker Compose:

```
  node-exporter:  
    container_name: node-exporter  
    image: prom/node-exporter
```

```

volumes:
  - /proc:/host/proc:ro
  - /sys:/host/sys:ro
  - /:/rootfs:ro
  command: '-collector.procfs=/host/proc -collector.sysfs=/host/sys -'
  collector.filesystem.ignored-mount-
  points="^(/rootfs|/host|)/(sys|proc|dev|host|etc)(\$\$|/)"
  collector.filesystem.ignored-fs-
  types="^(sys|proc|auto|cgroup|devpts|ns|au|fuse|.lxc|mqueue)(fs|\$\$)''"
  expose:
    - 9100

```

Ponownie montowane są różne elementy systemu plików hosta. Tym razem nie montujemy folderu `/var/lib/docker/`, ponieważ nie używamy tego narzędzia do monitorowania kontenerów. Potrzebujemy informacji o obciążeniu procesora, pamięci operacyjnej i dysku hosta. Zastosowaliśmy polecenie dynamicznie konfigurujące narzędzie *node-exporter*, a więc nie musimy przygotowywać i montować wewnątrz kontenera plików definiujących konfigurację.

Port narzędzia *node-exporter* jest udostępniany tylko usługom zdefiniowanym w pliku Docker Compose. Narzędzie to, podobnie jak usługa *cadvisor*, jest tylko węzłem końcowym, który wyświetla parametry hosta Docker pod adresem `http://node-exporter:9100/metrics/`.

Narzędzia *cadvisor* i *node-exporter* wspomagają pracę kolejnej usługi. Usługa *prometheus* wykonuje najczęstsze operacje. Jest to otwarte narzędzie przeznaczone do monitorowania opracowane przez autorów serwisu SoundCloud (<https://soundcloud.com/>).

```

prometheus:
  image: prom/prometheus
  container_name: prometheus
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
    - prometheus_data:/prometheus
  restart: unless-stopped
  expose:
    - 9090
  depends_on:
    - cAdvisor
    - node-exporter

```

Z przedstawionej definicji usługi wynika, że montujemy w jej kontenerze plik konfiguracyjny, a także wolumen o nazwie *prometheus_data*. Wspomniany plik konfiguracyjny zawiera dane określające źródła danych, które chcemy pobierać:

```

global:
  scrape_interval: 15s
  evaluation_interval: 15s
  external_labels:
    monitor: 'monitoring'

rule_files:

```

```
scrape_configs:  
  - job_name: 'prometheus'  
    static_configs:  
      - targets: ['localhost:9090']  
  
  - job_name: 'cadvisor'  
    static_configs:  
      - targets: ['cadvisor:8080']  
  
  - job_name: 'node-exporter'  
    static_configs:  
      - targets: ['node-exporter:9100']
```

Narzędzie Prometheus ma pobierać dane z węzłów końcowych co 15 sekund. Węzły te są zdefiniowane w sekcji `scrape_configs`. Znajazły się tam węzły usług `cadvisor` i `node-exporter`, a także węzeł definiowanej właśnie usługi `prometheus`. Wolumen `prometheus_data` jest tworzony w celu umożliwienia zapisu w bezpiecznym miejscu danych zebranych przez narzędzie Prometheus.

Głównym elementem narzędzia Prometheus jest baza danych szeregow czasowych. Narzędzie to pobiera dane, przetwarza je w celu ustalenia nazw i wartości parametrów kontenerów, a następnie zapisuje w swojej bazie wraz ze znacznikiem czasu.

Więcej informacji na temat modelu danych zastosowanego w narzędziu `prometheus` znajdziesz na stronie https://prometheus.io/docs/concepts/data_model/.

Narzędzie Prometheus jest również wyposażone w mechanizm wyszukiwania i interfejs programistyczny, a więc baza danych tego narzędzia doskonale nadaje się do przechowywania danych, które chcemy w niej gromadzić. Narzędzie to może generować podstawowe wykresy, ale w celu uzyskania wizualizacji zaleca się użycie narzędzia Grafana, które będzie naszą ostatnią usługą (jest to jedyna usługa, którą udostępnimy w sieci publicznej).

Grafana to otwarte narzędzie przeznaczone do tworzenia graficznych interpretacji baz danych szeregow czasowych, takich jak Graphite (<https://graphiteapp.org/>), InfluxDB (<https://www.influxdata.com/>) oraz Prometheus. Inne bazy danych są obsługiwane za pomocą odpowiednich wtyczek.

Oto definicja usługi `grafana` znajdująca się w pliku Docker Compose:

```
grafana:  
  image: grafana/grafana  
  container_name: grafana  
  volumes:  
    - grafana_data:/var/lib/grafana  
  env_file:
```

```

    - grafana.config
restart: unless-stopped
ports:
    - 3000:3000
depends_on:
    - prometheus

```

Wewnętrzną bazę danych narzędzia Grafana zapisujemy w wolumenie *grafana_data*. Zmienne środowiskowe tym razem nie są umieszczane w pliku Docker Compose. Są one ładowane z zewnętrznego pliku o nazwie *grafana.config*.

Oto zmienne zdefiniowane w tym pliku:

```

GF_SECURITY_ADMIN_USER=admin
GF_SECURITY_ADMIN_PASSWORD=password
GF_USERS_ALLOW_SIGN_UP=false

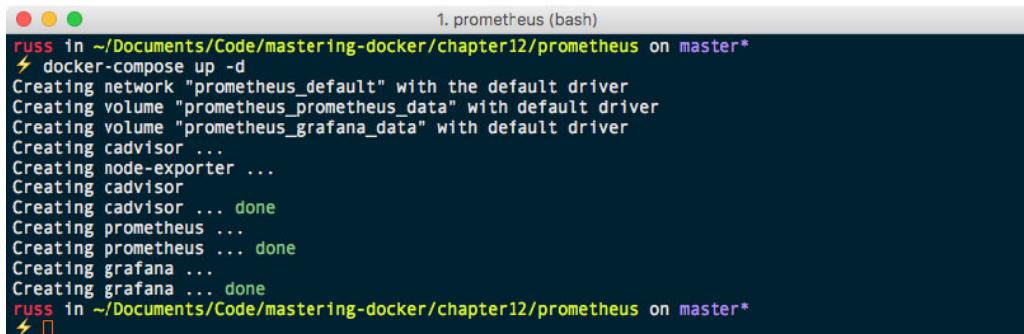
```

Zmienne te określają nazwę użytkownika i hasło. Umieszczenie ich w zewnętrznym pliku pozwala na zmodyfikowanie tych parametrów bez potrzeby wprowadzania zmian w głównym pliku Docker Compose.

Teraz już wiesz, do czego służy każda z czterech usług. Czas je uruchomić. W tym celu przejdź do folderu */r12/prometheus/* i uruchom w nim polecenie:

```
$ docker-compose up -d
```

Uruchomienie tego polecenia spowoduje utworzenie sieci i woluminów, a także pobranie obrazów z repozytorium Docker Hub. Na koniec uruchomione zostaną cztery usługi:



```

1. prometheus (bash)
russ in ~/Documents/Code/mastering-docker/chapter12/prometheus on master*
⚡ docker-compose up -d
Creating network "prometheus_default" with the default driver
Creating volume "prometheus_prometheus_data" with default driver
Creating volume "prometheus_grafana_data" with default driver
Creating cadvisor ...
Creating node-exporter ...
Creating cadvisor
Creating cadvisor ...
Creating prometheus ...
Creating prometheus ...
Creating grafana ...
Creating grafana ...
russ in ~/Documents/Code/mastering-docker/chapter12/prometheus on master*
⚡

```

Jeżeli od razu otworzysz w swojej przeglądarce stronę <http://localhost:3000/>, to jeszcze niczego na niej nie znajdziesz, ponieważ inicjalizacja narzędzia Grafana trwa kilka minut. Postęp procesu inicjalizacji możesz śledzić, przyglądając się komunikatom wyświetlonym po uruchomieniu polecenia:

```
$ docker-compose logs -f grafana
```

Po uruchomieniu tego polecenia w oknie Terminala pojawią się następujące komunikaty:

```
1. prometheus (docker-compose)
Set dashboard version to 1 where 0"
grafana    | t=2017-07-09T09:59:38+0000 lvl=info msg="Executing migration" logger=migrator id=
"save existing dashboard data in dashboard_version table v1"
grafana    | t=2017-07-09T09:59:39+0000 lvl=info msg="Created default admin user: admin"
grafana    | t=2017-07-09T09:59:39+0000 lvl=info msg="Starting plugin search" logger=plugins
grafana    | t=2017-07-09T09:59:39+0000 lvl=warn msg="Plugin dir does not exist" logger=plugin
s dir=/var/lib/grafana/plugins
grafana    | t=2017-07-09T09:59:39+0000 lvl=info msg="Plugin dir created" logger=plugins dir=/
var/lib/grafana/plugins
grafana    | t=2017-07-09T09:59:39+0000 lvl=info msg="Initializing Alerting" logger=alerting.e
ngine
grafana    | t=2017-07-09T09:59:39+0000 lvl=info msg="Initializing CleanUpService" logger=clea
nup
grafana    | t=2017-07-09T09:59:39+0000 lvl=info msg="Initializing Stream Manager"
grafana    | t=2017-07-09T09:59:39+0000 lvl=info msg="Initializing HTTP Server" logger=http.se
rver address=0.0.0.0:3000 protocol=http subUrl= socket=

```

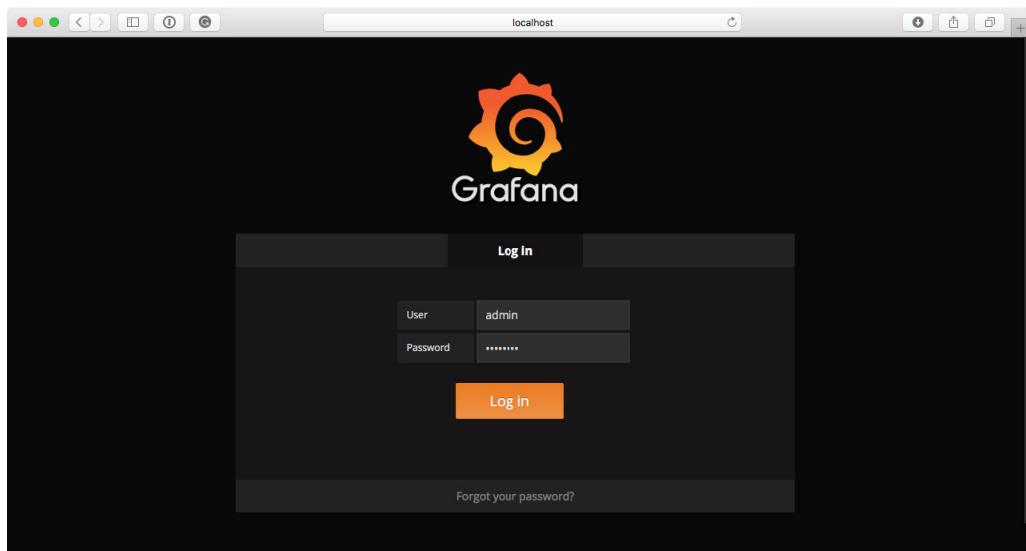
Narzędzie Grafana będzie dostępne, gdy na ekranie pojawi się komunikat *Initializing HTTP Server* (inicjalizacja serwera HTTP). Zanim zaczniemy z niego korzystać, musimy zrobić jeszcze jedną rzecz — skonfigurować źródło danych. Możemy zrobić to za pomocą przedstawionego niżej polecenia. Korzysta ono z interfejsu programistycznego narzędzia Grafana i można go używać tylko w systemach macOS i Linux:

```
$ curl 'http://admin:password@localhost:3000/api/datasources' -X POST -H 'Content-
➥Type: application/json; charset=UTF-8' --data-binary '{"name":"Prometheus","type":'
➥"prometheus","url":"http://prometheus:9090","access":"proxy","isDefault":true}'
```

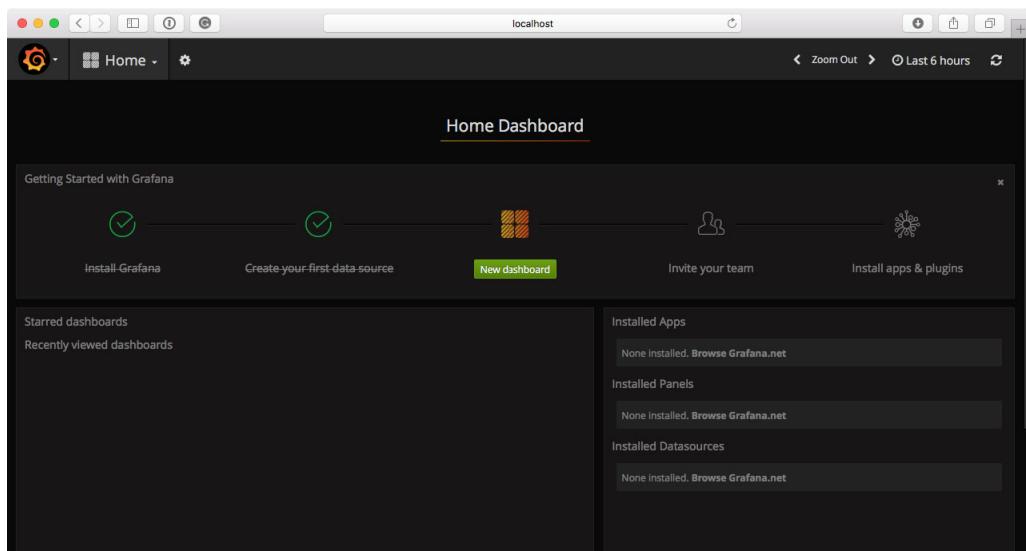
Polecenie to znajdziesz w pliku *CZYTAJ_TO.md* znajdującym się w archiwum pobranym ze strony <ftp://ftp.helion.pl/przykłady/dockaz.zip>, a więc nie musisz go przepisywać. Wystarczy posłużyć się metodą „kopiuj-wklej”.

```
1. prometheus (bash)
russ in ~/Documents/Code/mastering-docker/chapter12/prometheus on master*
⚡ curl 'http://admin:password@localhost:3000/api/datasources' -X POST -H 'Content-Type: application
➥/json; charset=UTF-8' --data-binary '{"name":"Prometheus","type":"prometheus","url":"http://prometheu
➥s:9090","access":"proxy","isDefault":true}'
{"id":1,"message":"Datasource added","name":"Prometheus"}russ in ~/Documents/Code/mastering-docker/c
hapter12/prometheus on master*
⚡
```

Nie martw się, jeżeli nie możesz skonfigurować źródła danych za pomocą interfejsu programistycznego. Źródło danych można skonfigurować po zalogowaniu się. Otwórz przeglądarkę i wpisz adres <http://localhost:3000/>. Na ekranie pojawi się ekran logowania:

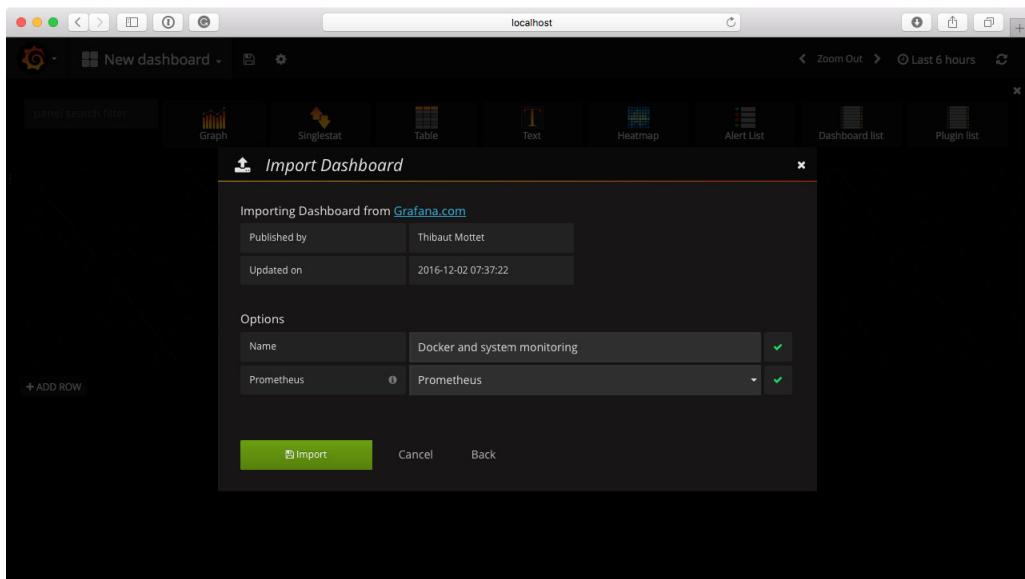


W polu *User* (użytkownik) wpisz nazwę `admin`, a w polu *Password* podaj hasło `password`. Po zalogowaniu się (jeżeli skonfigurowałeś wcześniej źródło danych) powinieneś zobaczyć następującą stronę:



Jeżeli nie udało Ci się wcześniej określić źródła danych, to zostaniesz o to poproszony. Wykonuj instrukcje wyświetlane na ekranie (w kolejnych krokach procesu konfiguracji możesz skorzystać z informacji zawartych w przedstawionym wcześniej poleceniu curl). Po skonfigurowaniu źródła danych na ekranie pojawi się strona *New Dashboard* (nowy ekran wskaźników).

Kliknij przycisk *Home* (dom) znajdujący się w lewym górnym rogu ekranu. Spowoduje to wyświetlenie menu. Po prawej stronie znajdziesz przycisk *Import Dashboard* (importuj ekran wskaźników). Kliknij go. Spowoduje to otwarcie strony, na której możesz załadować plik *json* (*Upload json File*), wprowadzić identyfikator lub adres ekranu wskaźników serwisu *Grafana.com* (*Grafana.com Dashboard*) albo wkleić kod JSON (*paste JSON*). W polu *Grafana.com Dashboard* wprowadź numer 893, a następnie kliknij przycisk *Load* (załaduj). Na ekranie pojawią się opcje importowania:



Wybierz źródło danych *Prometheus* i pozostaw domyślne ustawienia innych opcji. Na koniec kliknij przycisk *Import* (importuj).

Załadowany zostanie szablon *Docker and system monitoring* (monitorowanie Dockera i systemu) autorstwa Thibaut Mottet. Został on utworzony z myślą o usługach *cadvisor*, *node-exporter* i *Prometheus*. Zaimportowany szablon wygląda tak (patrz zrzut na następnej stronie).

Jak widzisz, dysponuję danymi z okresu dwóch godzin. Mogę je eksplorować, klikając różne grafy, a także zmieniając opcje wyświetlania znajdujące się w prawym górnym rogu ekranu. Więcej informacji na temat tego zestawu wskaźników znajdziesz na stronie <https://grafana.com/dashboards/893>.



Zgodnie z tym, co pisalem na początku tego podrozdziału, mamy do czynienia z bardzo rozbudowanym narzędziem. W przyszłości autorzy Dockera rozbudują wbudowany w platformę węzeł końcowy, tak aby przekazywał informacje nie tylko o silniku Docker Engine, ale również o samych kontenerach. Więcej informacji na temat wbudowanego węzła końcowego znajdziesz w oficjalnej dokumentacji Dockera na stronie <https://docs.docker.com/config/thirdparty/prometheus/>.

Istnieją również inne narzędzia przeznaczone do monitorowania środowiska Dockera. Większość z nich to niezależne oprogramowanie mające formę usług:

- **Sysdig Cloud:** <https://sysdig.com/>;
- **Datadog:** https://docs.datadoghq.com/integrations/docker_daemon/;
- **CoScale:** <http://www.coscale.com/docker-monitoring/>;
- **Dynatrace:** <https://www.dynatrace.com/capabilities/microservices-and-container-monitoring/>;
- **SignalFx:** <https://signalfx.com/docker-monitoring/>;
- **New Relic:** <https://newrelic.com/partner/docker>;
- **Sematext:** <https://sematext.com/docker/>.

Istnieją również rozwiązania dysponujące samodzielnym hostingiem:

- **Elastic Beats:** <https://www.elastic.co/products/beats>;
- **Sysdig:** <https://www.sysdig.org/>;
- **Zabbix:** <https://github.com/monitoringartist/zabbix-docker-monitoring>.

Jak widzisz, istnieje kilka solidnych narzędzi przeznaczonych do monitorowania środowiska Dockera. Być może korzystałś już wcześniej z jednego z nich. W takim przypadku łatwiej Ci będzie rozszerzać konfigurację Dockera, biorąc pod uwagę mechanizm monitorowania kontenerów.

Po zakończeniu przeglądania funkcji narzędzia Prometheus nie zapomnij o konieczności usunięcia jego kontenerów:

```
$ docker-compose down --volumes --rmi all
```

Polecenie to usunie wszystkie kontenery, wolumeny, obrazy i sieci.

Rozszerzanie na zewnętrzne platformy

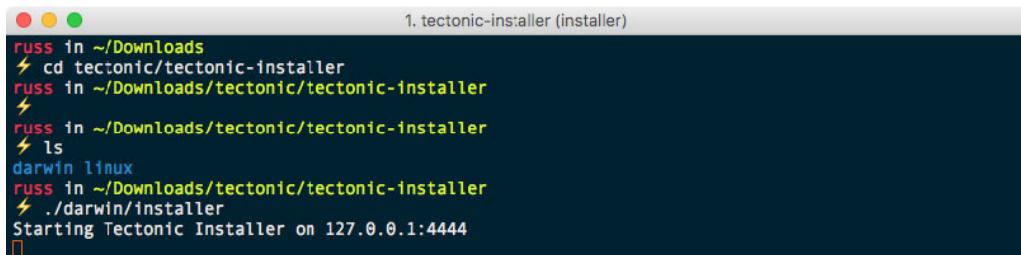
Przyjrzyliśmy się technikom rozszerzania środowiska Dockera na wybrane zewnętrzne platformy za pomocą narzędzi takich jak Docker Machine, Docker Swarm, Docker dla usług Amazon Web Services i Rancher — wiesz już, jak uruchomić hosty i klastry Dockera w publicznych chmurach, takich jak Amazon Web Services, Microsoft Azure i DigitalOcean.

W tym podrozdziale przyjrzymy się innym platformom obsługującym kontenery. Zaczniemy od systemu Kubernetes i instalatora Tectonic.

Instalator Tectonic

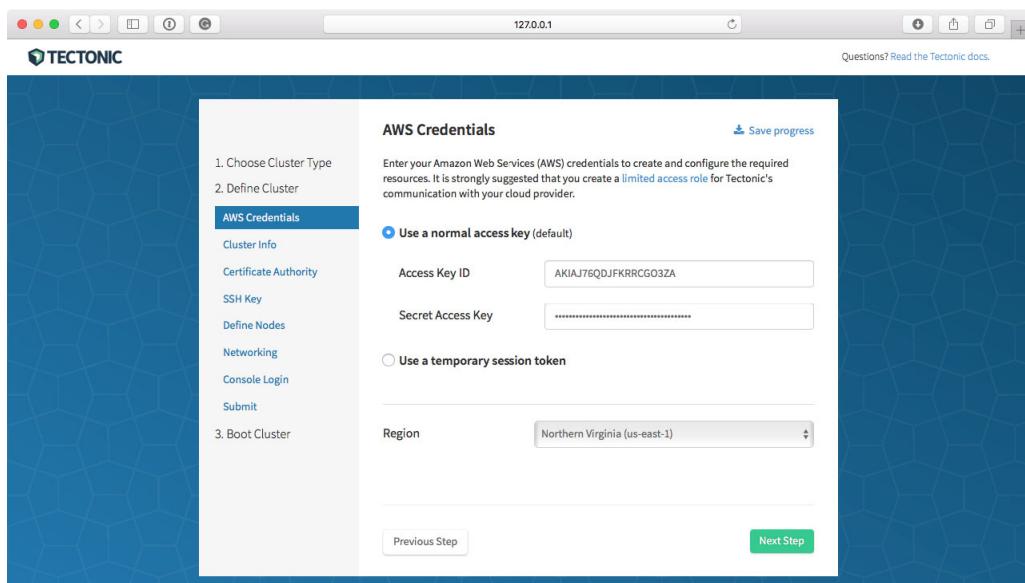
Tectonic firmy CoreOS to w pełni zarządzany instalator systemu Kubernetes. Doskonale nadaje się on do uruchamiania dużych klastrów Kubernetes i zarządzania nimi za pomocą centralnego, łatwego w obsłudze interfejsu. Jest to oprogramowanie płatne, ale umożliwia darmowe uruchamianie klastrów składających się maksymalnie z 10 węzłów. W celu założenia konta uprawniającego do pobrania instalatora Tectonic wejdź na stronę <https://coreos.com/tectonic/> i postępuj zgodnie z instrukcjami wyświetlonymi na ekranie.

Po założeniu konta i pobraniu instalatora musisz go uruchomić. Instalator tworzy lokalną usługę sieciową:



```
russ in ~/Downloads
$ cd tectonic/tectonic-installer
russ in ~/Downloads/tectonic/tectonic-installer
$ ls
darwin linux
russ in ~/Downloads/tectonic/tectonic-installer
$ ./darwin/installer
Starting Tectonic Installer on 127.0.0.1:4444
```

Instalator uruchamia się w oknie przeglądarki:



Instalator przeprowadzi Cię przez cały proces definiowania klastra Kubernetes. Podeczas swojej pierwszej instalacji warto pozostać przy domyślnych ustawieniach wielu opcji.

Po wprowadzeniu danych chmury, rozmiaru klastra, danych sieci i serwera DNS, kluczy SSH itd. instalator przygotuje, a następnie wdroży konfigurację narzędzia Terraform.

Terraform jest otwartym narzędziem orkiestracji utworzonym przez firmę HashiCorp. Pozwala ono na definiowanie infrastruktury za pomocą kodu. Narzędzie to współpracuje z większością usług chmury, obsługując w znacznym stopniu ich interfejsy programistyczne, co pozwala na korzystanie z praktycznie wszystkich oferowanych usług. Więcej informacji na ten temat znajdziesz na stronie <https://www.terraform.io/>.

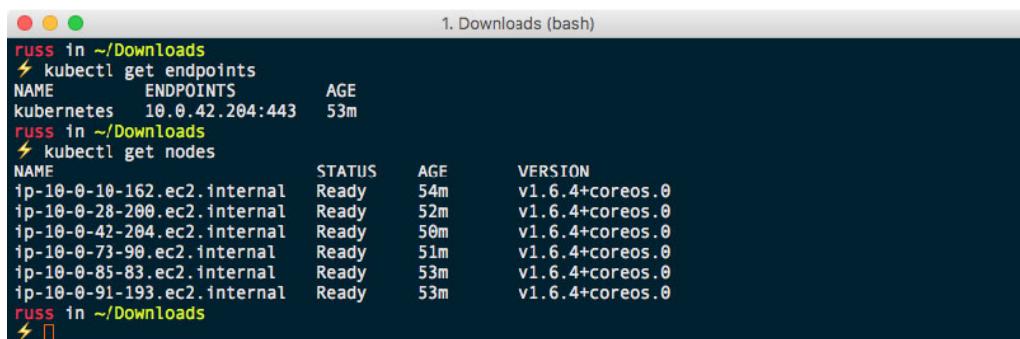
Na ekranie pojawiają się opcje umożliwiające pobranie kodu Terraform oraz innych zasobów wygenerowanych przez instalator Tectonic. Warto pobrać te wszystkie pliki i zapisać je w bezpiecznym miejscu. Pozwolą one później na odtworzenie klastra (patrz pierwszy zrzut na następnej stronie).

Instalacja trwa około kwadransa. Po jej zakończeniu możliwe będzie uzyskanie dostępu do konsoli Tectonic (*Tectonic Console*) (patrz drugi zrzut na następnej stronie).

The screenshot shows the Tectonic web interface with a blue hexagonal background. On the left, a sidebar lists steps: 1. Choose Cluster Type, 2. Define Cluster, 3. Boot Cluster. Step 3 is highlighted with a blue bar containing the text 'Start Installation'. Below the sidebar, it says 'Installation Complete'. The main area is titled 'Start Installation' with a 'Save progress' button. It displays a message: 'Kubernetes is starting up. We're committing your cluster details. Grab some tea and sit tight. This process can take up to 20 minutes. Status updates will appear below.' A green success message 'Terraform apply success' is shown with 'Show logs' and 'Save log' options. Below this, a box says 'Apply Succeeded.' with a note: 'If you've changed your mind, you can destroy your cluster.' It includes 'Destroy Cluster' and 'Retry Terraform Apply' buttons. A warning message in a box states: 'mckendrick.io's SOA TTL is 900 seconds and its SOA minimum TTL is 86400 seconds. The SOA record TTL and minimum TTL values determine how long to cache NVDOMAIN responses for. Installation cannot complete until matering-docker-k8s.mckendrick.io resolves. Read more here.' At the bottom, there are 'Start Over' and 'Download assets' buttons, and a 'Next Step' button.

The screenshot shows the Tectonic web interface with a blue hexagonal background. On the left, a sidebar lists categories: Workloads (Deployments, Replica Sets, Replication Controllers, Autoscalers), Daemon Sets (Jobs, Pods, Config Maps, Secrets), and Troubleshooting. The 'Cluster Status' page is the main content. It features a 'Cluster Health' section with two rows: 'Tectonic Console' (All systems go) and 'Kubernetes API Connection' (All systems go). To the right, a 'Software Details' table provides information: Kubernetes v1.6.4+coreos.0, Tectonic 1.6.4-tectonic.1 Release Notes, License 10 Nodes, Cloud Provider Amazon Web Services. On the right side, there are sections for 'Documentation' (Manage Your Account, End User Guide), 'Additional Support' (Full Documentation, Tectonic Forum, tectonic-feedback@coreos.com), and a footer with the text '© 2014 CoreOS Inc. All rights reserved.'

Konsola ta umożliwia zarządzanie systemem Kubernetes i pobranie pliku konfiguracyjnego `kubectl` umożliwiającego interakcję z klastrem Kubernetes z poziomu wiersza poleceń:



```
russ in ~/Downloads
⚡ kubectl get endpoints
NAME           ENDPOINTS      AGE
kubernetes     10.0.42.204:443  53m
russ in ~/Downloads
⚡ kubectl get nodes
NAME          STATUS    AGE     VERSION
ip-10-0-10-162.ec2.internal Ready   54m    v1.6.4+coreos.0
ip-10-0-28-200.ec2.internal  Ready   52m    v1.6.4+coreos.0
ip-10-0-42-204.ec2.internal Ready   50m    v1.6.4+coreos.0
ip-10-0-73-90.ec2.internal  Ready   51m    v1.6.4+coreos.0
ip-10-0-85-83.ec2.internal  Ready   53m    v1.6.4+coreos.0
ip-10-0-91-193.ec2.internal Ready   53m    v1.6.4+coreos.0
russ in ~/Downloads
⚡
```

Klastry Kubernetes mogą być używane jak wszystkie inne klastry. Przed wdrożeniem pierwszej aplikacji polecam Ci lekturę samouczka znajdującego się na stronie: <https://coreos.com/tectonic/docs/latest/tutorials/aws/first-app.html>.

Po więcej informacji na temat wdrażania klastrów Kubernetes przy użyciu instalatora Tectonic zajrzyj do poradnika użytkownika znajdującego się na stronie <https://coreos.com/tectonic/docs/latest/>. Opisano w nim wdrażanie na serwerach, a także w chmurach Amazon Web Services, Microsoft Azure i OpenStack.

Konfiguracja systemu Kubernetes jest bardzo skomplikowana. Stwierdzenie to dotyczy również utrzymania tego systemu. Uruchomienie prostego klastra nie jest niczym trudnym, ale skonfigurowanie automatycznego skalowania węzłów i wprowadzanie poprawek „w locie” może być bardzo kłopotliwe. Instalator Tectonic wykonuje za Ciebie te bardzo skomplikowane zadania, a przy tym pozwala dokładnie przyjrzeć się wykonywanym operacjom — umożliwia pobranie zasobów Terraform. Operacje te są przeprowadzane zgodnie z aktualnymi wytycznymi dotyczącymi dobrych praktyk pracy w systemie Kubernetes.

Jeżeli interesujesz się systemem Kubernetes, to polecam Ci przyjrzenie się instalatorowi Tectonic.

Platforma Heroku

Heroku różni się nieco od innych usług chmury — usługa ta jest uważana za **platformę będącą usługą** (*Platform as a Service, PaaS*). Zamiast wdrażać na niej kontenery, możesz je z nią powiązać. Platforma natomiast umożliwia uruchomienie usług takich jak PHP, Java, Node.js lub Python. W związku z tym na platformie Heroku możesz uruchomić aplikację Rails, a następnie dołączyć do niej kontener Dockera.

Dockera oraz platformy Heroku można używać razem — w tym celu należy utworzyć aplikację na platformie Heroku, a następnie skorzystać z kodu:

```
{  
    "name": "Nazwa aplikacji",  
    "description": "Aplikacja uruchamiająca kod w kontenerze Dockera",  
    "image": "<obraz_dockera>:<etykieta>",  
    "addons": [ "heroku-postgresql" ]  
}
```

Funkcja ta nie zadziała, jeżeli nie zainstalujesz wcześniej wtyczki platformy Heroku za pomocą polecenia:

```
$ heroku plugins:install heroku-docker
```

Zaprezentowany wcześniej kod może być użyty w kontekście wielu kompatybilnych z Heroku obrazów umieszczonych w repozytorium Docker Hub:

- *heroku/nodejs,*
- *heroku/ruby,*
- *heroku/jruby,*
- *heroku/python,*
- *heroku/scala,*
- *heroku/clojure,*
- *heroku/gradle,*
- *heroku/java,*
- *heroku/go,*
- *heroku/go-gb.*

Usługa Amazon Elastic Container Service

Amazon oferuje swoją własną usługę opartą na Dockerze. Nadano jej nazwę **Amazon Elastic Container Service** — w skrócie **Amazon ECS**. Podobnie jak jest w przypadku większości usług AWS, usługa Amazon ECS rozwija pełnię możliwości, gdy korzysta się z niej w połączeniu z innymi usługami AWS.

Korzystanie z samej usługi jest darmowe. Odpatnności są naliczane za zasoby używane przez kontenery. Inne usługi AWS używane w połączeniu z usługą Amazon ECS to:

- **Amazon Elastic Compute Cloud (EC2)** — usługa ta zapewnia moc obliczeniową (pamięć operacyjną, procesor i dostęp do sieci) hostom ECS. Bez tej usługi uruchomienie kontenerów jest niemożliwe.
- **Amazon Elastic Block Storage (EBS)** — usługa ta pozwala na przechowywanie zasobów obliczeniowych i kontenerów na poziomie bloku.
- **Elastic Load Balancing (ELB)** — mechanizmy równoważenia obciążzeń, które rozkładają obciążenia pomiędzy różnymi ośrodkami znajdującymi się w wybranym regionie.

- **Auto Scaling** — usługa ta umożliwia automatyczne skalowanie (w górę lub w dół) klastrów na podstawie określonych czynników, takich jak liczba hostów ECS, obciążenie procesora, liczba połączeń przychodzących do usługi ELB itd.
- **Virtual Private Cloud (VPC)** — usługa ta pozwala na tworzenie prywatnych sieci łączących wszystkie uruchomione usługi, dając przy tym ogromne możliwości zarządzania ruchem; umożliwia ona przeprowadzenie konfiguracji tabel trasowania, a także definiowanie uprawnień dostępu i zabezpieczeń.

Wszystkie te usługi w połączeniu z harmonogramem kontenerów Amazon pozwalają na szybkie tworzenie wysoce skalowalnych i optymalnych klastrów kontenerów, które mogą być zarządzane za pomocą interfejsu sieciowego Amazon ECS, narzędzi wiersza polecen chmury AWS lub interfejsu programistycznego.

Więcej informacji na temat usługi Amazon ECS znajdziesz na stronie <https://aws.amazon.com/ecs/>.

Jak wygląda środowisko produkcyjne?

W ostatnim podrozdziale opiszemy, jak powinno wyglądać środowisko produkcyjne. Dywagacje te będą krótsze, niż myślisz — nie możemy opisać wszystkich dostępnych opcji, ponieważ jest ich zbyt wiele. Ponadto po lekturze poprzednich rozdziałów powinieneś już wiedzieć, jakie rozwiązania sprawdzą się w przypadku Twoich potrzeb.

W kolejnych sekcjach opiszemy zagadnienia, które powinieneś rozważyć, planując swoje środowisko.

Hosty Dockera

Hosty Dockera to najważniejsze elementy środowiska. Bez nich uruchomienie kontenerów jest niemożliwe. Jak wiesz, uruchamiając hosty Dockera, należy rozważyć kilka rzeczy. Po pierwsze należy pamiętać o tym, że hosty, w których uruchomiono Dockera, nie powinny być używane do uruchamiania żadnych innych usług.

Mieszanie procesów

Nie warto podejmować ryzyka szybkiej instalacji Dockera na istniejącym hoście i uruchamiania w nim kontenera. Wiąże się to z ryzykiem w kwestii bezpieczeństwa, ponieważ w tym samym hoście mogą znaleźć się różne izolowane i nieizolowane procesy. Ponadto w takiej sytuacji mogą wystąpić problemy z wydajnością, ponieważ nie możesz nałożyć limitu zasobów użytkowanych przez aplikacje, które nie są uruchomione w kontenerze. Aplikacje takie mogą wpływać negatywnie na pracę kontenerów.

Wiele odizolowanych hostów Dockera

W jaki sposób obsłużysz kolejne hosty Dockera? Korzystanie z narzędzia takiego jak Portainer to wspaniale rozwiążanie, ale używanie go w celu zarządzania więcej niż kilkoma hostami może stać się kłopotliwe. Ponadto w przypadku uruchomienia wielu odizolowanych hostów Dockera nie można przenosić kontenerów pomiędzy hostami.

Oczywiście można w takim przypadku użyć narzędzia takiego jak np. Weave Net w celu utworzenia sieci kontenerów znajdujących się w różnych hostach. W zależności od środowiska hosta możesz dysponować również możliwością tworzenia zewnętrznych woluminów i udostępniania ich w razie konieczności hostom Dockera, ale proces migracji kontenerów pomiędzy hostami będzie najprawdopodobniej musiał być zarządzany ręcznie.

Kierowanie żądań do kontenerów

W przypadku korzystania z wielu hostów należy określić sposób rozdzielania żądań pomiędzy kontenerami.

W przypadku zastosowania zewnętrznego mechanizmu równoważącego obciążenie, takiego jak np. ELB chmury AWS, lub rozwiązania dedykowanego dla lokalnego klastra należy dysponować możliwością dynamicznego dodawania przekierowań ruchu przychodzącego do portu x mechanizmu równoważącego obciążenia do portów y hostów Dockera, które kierują ruch dalej do kontenerów.

Jeżeli dysponujesz wieloma kontenerami, które są dostępne na tym samym zewnętrznym porcie, to musisz określić sposób rozdzielania ruchu pomiędzy nimi.

Zamiast kierować ruch na podstawie portów, możesz również zainstalować serwer pośredniczący, taki jak **Traefik** (<https://traefik.io/>), **HAProxy** (<http://www.haproxy.org/>) lub **NGINX** (<https://nginx.org/>), który miałby przyjmować i kierować żądania dalej do hostów na podstawie domen lub subdomen.

Porty mogą zostać użyte do obsługi witryny — np. odpowiednio skonfigurowany kontener może obsługiwać cały ruch kierowany do portów 80 i 443. Korzystanie z mechanizmu rozdzielania ruchu pomiędzy wirtualnymi hostami pozwala na kierowanie ruchu *domena-a.com* do *kontenera a* i ruchu *domena-b.com* do *kontenera b*, a obie te domeny mogą prowadzić do tego samego adresu IP i portu.

Obsługa klastrów

Wiele problemów opisanych w poprzednim rozdziale można rozwiązać za pomocą narzędzi obsługujących klastry, takich jak Docker Swarm, Kubernetes lub Cattle.

Kompatybilność

Pomimo tego, że aplikacja działa poprawnie w lokalnym środowisku Dockera na komputerze programisty, należy zagwarantować, że po wdrożeniu jej np. w klastrze Kubernetes będzie ona działała tak samo.

Dziewięć na dziesięć wdrożeń przebiegne bezproblemowo, ale musisz zwracać uwagę na to, jak aplikacja komunikuje się wewnętrznie z innymi kontenerami, z którymi współpracuje.

Architektury referencyjne

Czy wybrana przez Ciebie technologia obsługi klastrów oferuje architektury referencyjne? Wdrażając kластer, zawsze warto sprawdzić, czy jest on tworzony zgodnie z dobrymi praktykami pracy w danym środowisku. Dzięki temu nie stworzysz jednej wielkiej porażki.

Jakie są zalecane zasoby? Nie ma sensu wdrażać klastra z pięcioma węzłami zarządzającymi i jednym hostem Dockera, tak samo jak bezsensowne jest wdrażanie pięciu hostów Dockera i jednego serwera zarządzającego.

Jakie technologie pomocnicze obsługuje wybrana przez Ciebie technologia klastrów? Czy obsługuje zdalne tworzenie magazynów, mechanizmy równoważenia obciążenia i zapory połączeń?

Komunikacja klastra

Jakie są wymagania dotyczące komunikacji pomiędzy klastrem i hostami zarządzającymi? Czy przesył danych przez kластer powinien odbywać się w oddzielnej, odizolowanej sieci?

Czy dostęp do elementu klastra może być ograniczony? Czy komunikacja może zostać zaszyfrowana? Jakie informacje o klastrze mogą zostać udostępnione? Czy to nie zwiększa zagrożenia atakiem hakerskim?

Do jakich zewnętrznych interfejsów (np. chmur) kластer musi uzyskać dostęp? Jak bezpieczne jest przechowywanie danych logowania do tych interfejsów?

Rejestry obrazów

Jaką postać ma aplikacja? Czy jej kod umieszczasz w formie obrazu? Jeżeli tak, to czy potrzebujesz prywatnego rejestru obrazów? Czy wystarczą Ci możliwości oferowane przez usługi takie jak Docker Hub, **Docker Trusted Registry (DTR)** lub Quay?

Jeżeli musisz korzystać z prywatnego rejestru, to w jakim miejscu środowiska należy go umieścić? Kto powinien mieć do niego dostęp? Czy można go połączyć z usługą katalogową taką jak np. Active Directory?

Podsumowanie

W tym rozdziale przyjrzyliśmy się przepływom zadań w platformie Docker, a także zagadniom związanym z monitorowaniem kontenerów i hostów Dockera.

Opisaliśmy trzy zewnętrzne platformy i zebraliśmy pytania, na które powinieneś odpowiedzieć, definiując środowisko produkcyjne. Na pytania te nie ma jednej poprawnej odpowiedzi — odpowiadając na nie, należy brać pod uwagę własne wymagania dotyczące środowiska Dockera.

Najlepiej jest opracować koncepcję środowiska wdrożeniowego, a następnie pomyśleć o wszystkich możliwych scenariuszach katastrof, do których może w nim dojść. Szukając dobrej architektury referencyjnej, możesz przyznać się usługom takim jak Tectonic i Amazon ECS — pozwoli to na przyśpieszenie pracy i ograniczenie błądzenia metodą prób i błędów.

W kolejnym rozdziale przyjrzymy się przenoszeniu odkrywania usług z poziomu kontenera na poziom aplikacji, a także poruszmy zagadnienia związane z dalszym rozwojem umiejętności obsługi platformy Docker.

Dalsze kroki z Dockerem

To już ostatni rozdział tej książki! Opiszemy w nim zagadnienia związane z wykrywaniem usług przez aplikacje uruchomione w środowisku kontenerów. Ponadto przeanalizujemy kierunki dalszego rozwoju platformy Docker. Dowiesz się, jak możesz wnieść wkład w rozwój tej technologii.

Wykrywanie usług

Dotychczas uruchamialiśmy aplikacje, które nie musiały wiedzieć, w jakim środowisku pracowały, ponieważ przepływem danych sterowaliśmy z poziomu sieci. A gdyby aplikacja musiała znać lokalizację pozostałych kontenerów?

Consul

Consul jest otwartym silnikiem odkrywania usług opracowanym przez firmę Hashicorp (<https://www.hashicorp.com/>). Jest to jedno z bardziej popularnych rozwiązań tego typu (jest ono popularne również poza światem kontenerów Docker). Jego cztery główne funkcje to:

- odkrywanie usług;
- przechowywanie par klucz-wartość;
- sprawdzanie statusu;
- możliwość współpracy z wieloma centrami danych.

Szczegółowo przyjrzymy się tylko funkcji odkrywania usług, ale wspomnimy również o zastosowaniach innych usług narzędzia Consul.

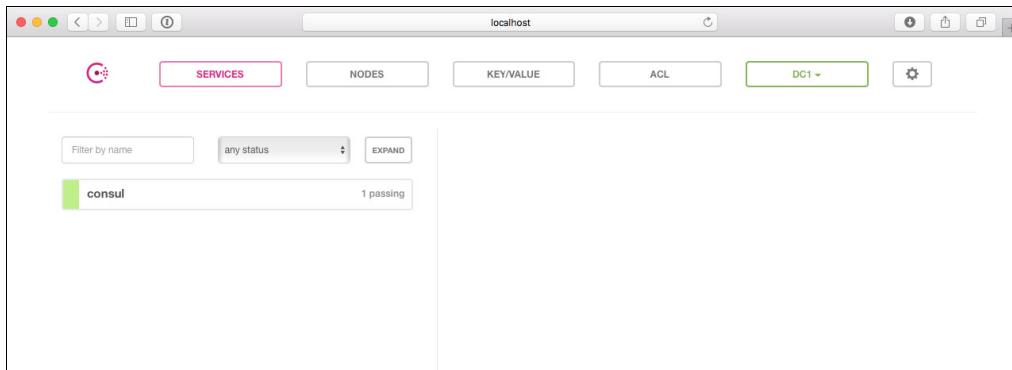
Uruchamianie narzędzia Consul

Zacznijmy od uruchomienia narzędzia Consul. Skorzystamy podczas tej operacji z oficjalnego obrazu narzędzia Consul pobranego z serwisu Docker Hub. Przedstawiony tutaj proces instalacji nie będzie zapewniał wysokiej dostępności narzędzia Consul, ale narzędzie to można skonfigurować w celu zapewnienia takiej dostępności.

Przed uruchomieniem kontenera Consul należy utworzyć sieć. W tym celu uruchomimy następujące polecenia:

```
$ docker network create consul
$ docker container run -d --name=consul --network=consul -p 8500:8500 -p
8600:8600/udp consul
```

Utworzyliśmy sieć o nazwie consul i kontener z otwartymi portami numer 8500 i 8600. W dalszej części tego podrozdziału przyjrzymy się portowi nr 8600. Na razie otwórz przeglądarkę i wejdź na stronę <http://localhost:8500/>. Na ekranie pojawi się graficzny interfejs narzędzia Consul, który został uruchomiony na porcie nr 8500. Strona wyświetlana w przeglądarce wygląda tak:



Jak widzisz, niewiele dzieje się w narzędziu. Dysponujemy jedną zarejestrowaną usługą (samym narzędziem Consul).

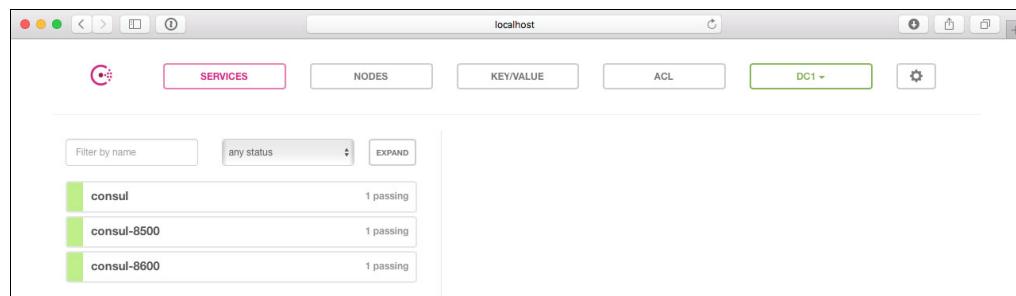
Uruchamianie usługi Registrar

Consul od razu po uruchomieniu nie widzi innych kontenerów, a więc musimy utworzyć żądanie HTTP kierowane do tego narzędzia podczas uruchamiania każdego kontenera. Można to zrobić za pomocą usługi **Registrar** opracowanej przez Glider Labs (<http://gliderlabs.github.io/registrator/latest/>). Usługę tę można uruchomić za pomocą polecenia:

```
$ docker container run -d \
--name=registrator --network=consul \
--volume=/var/run/docker.sock:/tmp/docker.sock \
gliderlabs/registrator:latest consul://consul:8500
```

Z polecenia tego wynika, że w kontenerze usługi Registrar zamontowano plik gniazda Docker Engine. Dzięki temu kontener ten ma bezpośredni dostęp do silnika Docker Engine uruchomionego w hoście, a więc ma stały dostęp do informacji o wprowadzanych zmianach, takich jak np. uruchamianie nowych kontenerów. Usługa Registrar została uruchomiona z parametrem `consul://consul:8500`. Jest to parametr konfiguracyjny określający adres narzędzia Consul. Nie musimy przeprowadzać żadnych innych czynności konfiguracyjnych. Kontenery usług Registrar i Consul są uruchomione w tej samej sieci, a więc wszystkie żądania będą kierowane na podstawie nazwy i portu docelowego kontenera (w naszym przypadku nazwa kontenera to `consul`, a numer portu to 8500).

Wróćmy do okna przeglądarki. Powinny pojawić się w nim dwie dodatkowe usługi:



Kontener narzędzia Consul został nazwany `consul` i udostępnia on porty nr 8500 i 8600, a więc narzędzie Registrar dodało te dwa wpisy do listy. Po zarejestrowaniu nowych kontenerów możemy uruchomić kontener testowy. Podobnie jak w poprzednich rozdziałach będziemy korzystać z aplikacji Cluster, ale tym razem nie będziemy zaprzątać sobie głowy uzyskiwaniem dostępu do samej aplikacji.

Kierowanie zapytań do narzędzia Consul

W celu uruchomienia kontenera aplikacji skorzystaj z następującego polecenia:

```
$ docker container run -d -p 80 -l "SERVICE_NAME=cluster" -l "SERVICE_TAGS=app,web"
→russmckendrick/cluster
```

W poleceniu dodaliśmy dwie etykiety za pomocą flagi `-l`. Etykiety te przekazują metadane do usługi Registrar, która przekazuje je dalej — do narzędzia Consul. Dodaliśmy następujące etykiety:

- `SERVICE_NAME` — nazwy usług umożliwiają grupowanie kontenerów.
- `SERVICE_TAGS` — wyróżniki kontenerów przydają się podczas identyfikowania usługi.

Na liście w oknie przeglądarki powinna pojawić się usługa o nazwie `cluster`. Wybierz ją w celu wyświetlenia informacji o niej:

The screenshot shows the Consul UI interface. At the top, there are tabs for SERVICES, NODES, KEY/VALUE, and ACL, with 'SERVICES' being the active tab. Below the tabs, there are filters for 'Filter by name' and 'any status'. A 'cluster' service is selected, indicated by a pink bar. The cluster has four services listed: 'consul' (status: 1 passing), 'consul-8500' (status: 1 passing), 'consul-8600' (status: 1 passing), and 'cluster' (status: 1 passing). On the right side, under 'cluster', there are sections for 'TAGS' (app, web) and 'NODES'. One node is listed: '230667de6b6c' (IP: 127.0.0.1, status: 1 passing). Below the node, it says 'Serf Health Status' and 'serfHealth'.

W oknie widoczne są zdefiniowane przez nas wyróżniki, a także informacja o statusie (*I passing*). Status usługi można sprawdzić również za pomocą interfejsu programistycznego wbudowanego w narzędzie Consul. W tym celu należy skorzystać z polecenia:

```
$ curl http://localhost:8500/v1/catalog/service/cluster/
```

Po jego uruchomieniu zwrócona zostanie tabela JSON zawierająca informacje o usłudze, mająca następującą formę:

```
[  
  {  
    "ID": "c6aeffbb-a64d-1d54-e335-f339e692dc71",  
    "Node": "230667de6b6c",  
    "Address": "127.0.0.1",  
    "Datacenter": "dc1",  
    "TaggedAddresses": {  
      "lan": "127.0.0.1",  
      "wan": "127.0.0.1"  
    },  
    "NodeMeta": {},  
    "ServiceID": "fd2a45346a81:peaceful_shirley:80",  
    "ServiceName": "cluster",  
    "ServiceTags": [  
      "app",  
      "web"  
    ],  
    "ServiceAddress": "",  
    "ServicePort": 32790,  
    "ServiceEnableTagOverride": false,  
    "CreateIndex": 184,  
    "ModifyIndex": 184  
  }]  
]
```

Jak widzisz, wśród zwróconych danych znajduje się kilka cennych informacji opisujących pracę aktywnej usługi.

Możemy odczytać:

- adres IP kontenera hosta usługi (**Address**);
- identyfikator UUID, nazwę i otwarty port kontenera (**ServiceID**);
- nazwę usługi (**ServiceName**);
- zdefiniowane wyróżniki usługi (**ServiceTags**).

Ponadto w zwróconych danych znajduje się wiele innych informacji o samej usłudze — np. udostępniony port hosta (**ServicePort**).

Narzędzie Consul poza interfejsem sieciowym obsługuje również zapytania kierowane do jego bazy za pośrednictwem usługi DNS.

Zapytania do wbudowanej w narzędzie Consul usługi DNS możemy kierować do portu nr 8600, który został otwarty w momencie uruchamiania kontenera Consul dla protokołu UDP. W celu skierowania zapytania należy uruchomić następujące polecenie:

```
$ dig @127.0.0.1 -p8600 cluster.service.consul
```

Polecenie to zwróci dane w następującej formie:

```
; <>> DiG 9.8.3-P1 <>> @127.0.0.1 -p8600 cluster.service.consul
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43651
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;cluster.service.consul. IN

;; ANSWER SECTION:
cluster.service.consul. 0 IN A 127.0.0.1

;; Query time: 1 msec
;; SERVER: 127.0.0.1#8600(127.0.0.1)
;; WHEN: Mon May 29 18:21:23 2017
;; MSG SIZE rcvd: 56
```

Nasz przykład jest skonfigurowany tak, aby korzystał z hosta lokalnego znajdującego się pod adresem 127.0.0.1. Zwrócone informacje nie są zbyt interesujące, ale możemy uzyskać bardziej szczegółowe informacje za pomocą polecenia:

```
$ dig @127.0.0.1 -p8600 cluster.service.consul SRV
```

Dodanie argumentu SRV na końcu polecenia zwróciło rekord SRV, który zawiera więcej informacji na temat użytych portów (w moim przypadku są to informacje o porcie nr 32790):

```
; <>> DiG 9.8.3-P1 <>> @127.0.0.1 -p8600 cluster.service.consul SRV
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21185
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;cluster.service.consul. IN SRV

;; ANSWER SECTION:
cluster.service.consul. 0 IN SRV 1 1 32790 230667de6b6c.node.dc1.consul.

;; ADDITIONAL SECTION:
230667de6b6c.node.dc1.consul. 0 IN A 127.0.0.1

;; Query time: 1 msec
;; SERVER: 127.0.0.1#8600(127.0.0.1)
;; WHEN: Mon May 29 18:25:44 2017
;; MSG SIZE rcvd: 98
```

Skorzystajmy dwukrotnie z poniższego polecenia (spowoduje to uruchomienie dwóch kolejnych kontenerów):

```
$ docker container run -d -p 80 -l "SERVICE_NAME=cluster" -l "SERVICE_TAGS=app,web"
↳russmckendrick/cluster
```

Teraz zwrócone zostaną nieco inne dane:

```
; <>> DiG 9.8.3-P1 <>> @127.0.0.1 -p8600 cluster.service.consul SRV
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 51192
;; flags: qr aa rd; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 3
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;cluster.service.consul. IN SRV

;; ANSWER SECTION:
cluster.service.consul. 0 IN SRV 1 1 32792 230667de6b6c.node.dc1.consul.
cluster.service.consul. 0 IN SRV 1 1 32790 230667de6b6c.node.dc1.consul.
cluster.service.consul. 0 IN SRV 1 1 32791 230667de6b6c.node.dc1.consul.

;; ADDITIONAL SECTION:
230667de6b6c.node.dc1.consul. 0 IN A 127.0.0.1
230667de6b6c.node.dc1.consul. 0 IN A 127.0.0.1
230667de6b6c.node.dc1.consul. 0 IN A 127.0.0.1

;; Query time: 1 msec
;; SERVER: 127.0.0.1#8600(127.0.0.1)
;; WHEN: Mon May 29 18:28:54 2017
;; MSG SIZE rcvd: 170
```

Zwracane dane zawierają informacje na temat wszystkich trzech aktywnych kontenerów, które zostały uruchomione w usłudze cluster. Informacje na temat trzech kontenerów są również wyświetlane w interfejsie graficznym:

The screenshot shows the Consul UI interface. At the top, there are tabs: SERVICES (highlighted in pink), NODES, KEY/VALUE, ACL, and DC1 (highlighted in green). Below the tabs, there are filters: 'Filter by name' (empty), 'any status' (selected), and 'EXPAND'. The main area is titled 'cluster' and shows the following data:

- TAGS:** app, web
- NODES:**
 - 230667de6b6c 127.0.0.1 (1 passing) - Serf Health Status: passing
 - 230667de6b6c 127.0.0.1 (1 passing) - Serf Health Status: passing
 - 230667de6b6c 127.0.0.1 (1 passing) - Serf Health Status: passing

Istnieje możliwość wyszukiwania w bazie określonych wyróżników. Spróbujmy uruchomić czwarty kontener, ale tym razem wyróżnik app zmieńmy na dev:

```
$ docker container run -d -p 80 -l "SERVICE_NAME=cluster" -l "SERVICE_TAGS=dev,web"
→russmckendrick/cluster
```

W celu znalezienia w bazie wyróżnika dev należy uruchomić następujące polecenie:

```
$ curl http://localhost:8500/v1/catalog/service/cluster?tag=dev
```

W wyświetlonych danych powinien znaleźć się tylko jeden kontener. Te same informacje możemy uzyskać za pomocą polecenia `dig @127.0.0.1 -p8600 dev.cluster.service.consul SRV`

```
$ dig @127.0.0.1 -p8600 dev.cluster.service.consul SRV
```

Dotychczas korzystaliśmy z wbudowanego interfejsu narzędzia Consul i usługi DNS, chcąc uzyskać informacje na temat kontenerów. Jeżeli chcesz pobierać dane na temat kontenerów hosta zebrane za pomocą narzędzia Consul do własnej aplikacji, to najprawdopodobniej będziesz to robić właśnie za pomocą interfejsu programistycznego, ale w przypadku potrzeby ustalenia pewnych informacji o kontenerach w celu aktualizowania pliku konfiguracyjnego zwykle nie warto pisać własnego kodu przetwarzającego dane dostępne w narzędziu Consul.

Szablony narzędzia Consul

Firma Hashicorp opracowała również narzędzie pomocnicze umożliwiające podłączenie się do zainstalowanej aplikacji Consul i przeprowadzenie operacji parsowania szablonu — wypełnienia go pobranymi danymi. Narzędziu temu nadano nazwę Consul Template. Jeżeli korzystasz z systemu macOS z menedżerem Homebrew, to możesz je zainstalować za pomocą polecenia:

```
brew install consul-template
```

Jeżeli nie posiadasz menedżera Homebrew, to musisz skorzystać z instrukcji instalacji znajdujących się na stronie projektu w repozytorium GitHub: <https://github.com/hashicorp/consul-template/>.

Utwórzmy prosty szablon — spróbujmy umieścić adres i port każdego z kontenerów naszej usługi o nazwie *cluster* w minimalistycznym pliku konfiguracyjnym. Oto szablon zapisany w pliku */tmp/consul.ctmpl*:

```
$ echo '${{range service "cluster"}} server {{.Address}}:{{.Port}}\n{{end}}' > /tmp/consul.ctmpl
```

Szablony narzędzia Consul Template są zapisywane w formacie Go Template. Więcej informacji na jego temat znajdziesz na stronie <https://golang.org/pkg/text/template/>.

Z zaprezentowanego szablonu można wywnioskować, że kierujemy zapytanie `{}range service "cluster"{}` do usługi o nazwie *cluster*. Słowo *server* nie zostało ujęte w nawiasy klamrowe, więc jest ono po prostu wyświetlane. W tym miejscu podczas generowania zapytania zostanie umieszczony adres IP serwera. Kod `{{.Address}}` służy do uzyskania adresu, a zapis `{{.Port}}` służy do ustalenia portu. Zwróć uwagę, że w poleceniu umieszczono również dwukropki : i znak końca wiersza \n. Na końcu zapytania umieszczono zamkający zapis `{}end{}` (został on umieszczony po pobraniu wszystkich informacji).

Mamy już szablon, a więc możemy uruchomić polecenie generujące zapytanie o informacje dotyczące usługi i zapisujące jego wynik w pliku o nazwie */tmp/consul.result*:

```
$ consul-template -once -consul-addr 127.0.0.1:8500 -template /tmp/consul.ctmpl:/tmp/consul.result
```

We wspomnianym pliku znajdą się dane sformatowane w następujący sposób:

```
server 127.0.0.1:32792  
server 127.0.0.1:32790  
server 127.0.0.1:32791  
server 127.0.0.1:32793
```

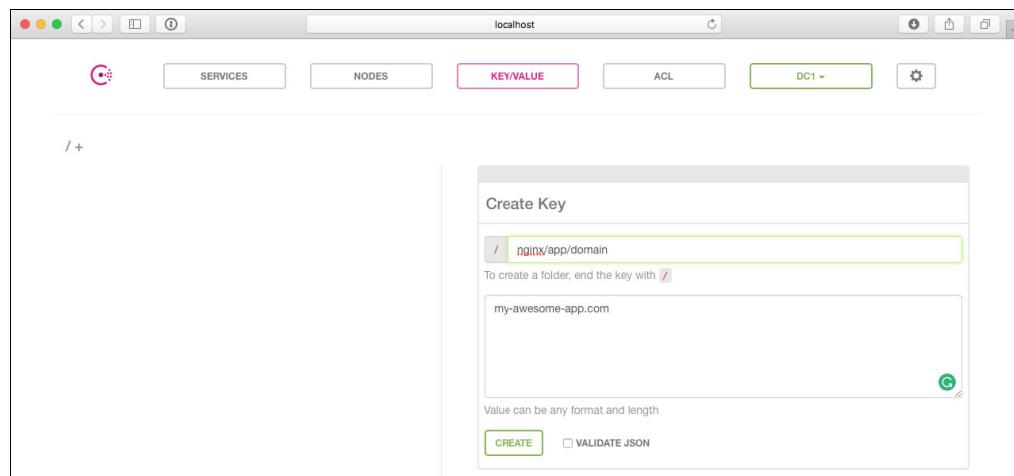
W poleceniu uruchomionym w celu wygenerowania szablonu użyto flagi `-once`. Flaga ta sprawiła, że narzędzie Consul Template zostało uruchomione jednokrotnie, a następnie zamknięte. Bez flagi `-once` narzędzie to byłoby uruchomione przy wprowadzeniu każdej modyfikacji

usługi `cluster` i generowałoby nowy plik szablonu. Użycie flagi `-exec` sprawiłoby, że polece-
nie byłoby uruchamiane przy każdym ponownym zapisaniu pliku konfiguracyjnego. Spróbuj
uruchomić poniższe przykładowe polecenie:

```
$ consul-template -consul-addr 127.0.0.1:8500 \
    -template /tmp/nginx.ctmpl:/etc/nginx/conf.d/default.conf \
    -exec "systemctl nginx reload"
```

Polecenie to wygeneruje plik konfiguracji usługi NGINX i będzie przeładowywało tę usługę
za każdym razem, gdy zostanie do niej dodany kontener lub gdy jej kontener zostanie usunięty.

Sekcja `KEY/VALUE` (klucz-wartość) narzędzia Consul może przydać się do sortowania ele-
mentów konfiguracji usługi NGINX:



Zbiór par kluczy i wartości można traktować jak bazę danych zmiennych środowiskowych.
Podczas uruchamiania kontenerów można zdefiniować parę `DOMAIN_NAME = www.domena.com` —
kluczowi `DOMAIN_NAME` (nazwa domeny) przypisać wartość `www.domain.com`.

Zaletą korzystania ze zbioru par klucz-wartość jest to, że informację tę można uzyskać za po-
mocą interfejsu programistycznego i polecenia:

```
$ curl http://localhost:8500/v1/kv/nginx/app/domain?raw
```

Można ją również uzyskać za pomocą narzędzia Consul Template:

```
$ echo '${{range service "cluster"}}{{ key "nginx/app/domain" }}\n{{.Address}}:{{.Port}}\n{{end}}' > /tmp/consul.ctmpl
$ consul-template -consul-addr 127.0.0.1:8500 -template
/tmp/consul.ctmpl:/tmp/consul.result
```

Po uruchomieniu polecenia zwrócone zostaną następujące dane:

```
moja-cudowna-aplikacja.com 127.0.0.1:32792  
moja-cudowna-aplikacja.com 127.0.0.1:32790  
moja-cudowna-aplikacja.com 127.0.0.1:32791  
moja-cudowna-aplikacja.com 127.0.0.1:32793
```

Wysoka dostępność narzędzia Consul

Przypominam, że nie polecam uruchamiania narzędzia Consul w jednowzłowej konfiguracji środowiska Docker. Jedną z głównych cech tego narzędzia jest to, że może ono działać w wysoce dostępnych klastrach. Więcej informacji na temat pracy z tego typu klastrami znajdziesz w sekcji *Client Mode* strony narzędzia Consul w sklepie Docker Store (<https://store.docker.com/images/consul/>) lub w poradniku dla poczatkujacych uzytkownikow tego narzędzia znajdujacym sie na stronie <https://www.consul.io/intro/getting-started/install.html>.

Narzędzie etcd

Narzędzie etcd jest alternatywą dla aplikacji Consul. Jeżeli chcesz, aby środowiska Twoich hostów były bardzo lekkie, i wolisz korzystać z systemu CoreOS, to powinieneś zapoznać się z możliwościami oferowanymi przez etcd, ponieważ jest to domyślne narzędzie odkrywania usług tego systemu. Odkrywanie jest wykonywane za pomocą dynamicznego rejestru konfiguracji. Po skonfigurowaniu narzędzia etcd w każdym hoście CoreOS możliwa jest dystrybucja i kopiowanie par klucz-wartość, co umożliwia odkrywanie hostów utworzonych przez siebie, a także odkrywanie nowych hostów.

Więcej informacji na temat narzędzia etcd znajdziesz na stronie https://en.wikipedia.org/wiki/Container_Linux_by_CoreOS#ETCD. Warto również zajrzeć na stronę <https://github.com/coreos/etcd> — znajdziesz na niej informacje na temat funkcji narzędzia etcd, rozwiązywania potencjalnych problemów, a także wskazówki, adresy list mailingowych i zgłoszone błędy. Możesz również wejść na stronę <https://coreos.com/etcd/>.

Narzędzie Registrar obsługuje etcd jako swoje zaplecze, a więc jeżeli narzędzie etcd zostało uruchomione w ramach środowiska CoreOs lub Kubernetes, to łatwo będzie je połączyć z własnymi usługami.

Interfejs Docker API

Oczywiście nikt nie zabrania Ci korzystać z interfejsu programistycznego Dockera, ale jest on dość skomplikowany i bardzo łatwo o przygniecenie nawalem zwracanych informacji. Przyjrzymy się przykładowi uruchamiania prostego, pojedynczego kontenera:

```
$ docker container run -d -p 80 russmckendrick/cluster
```

Teraz w hoście pracuje tylko jeden kontener. Uruchomienie poniższego polecenia spowoduje wysłanie zapytania do interfejsu programistycznego Dockera i wyświetlenie informacji o wszystkich aktywnych kontenerach w postaci tablicy JSON:

```
$ curl --unix-socket /var/run/docker.sock "http://v1.27/containers/json" | python -m
→json.tool
```

W wyniku uruchomienia tego polecenia zwrócona zostanie następująca tablica:

```
[
  {
    "Command": "supervisord -c /etc/supervisord.conf",
    "Created": 1496083791,
    "HostConfig": {
      "NetworkMode": "default"
    },
    "Id": "a4c2c76ace78e25e1c42833967447042f8607c453c14f443b730f605f2e6 5039",
    "Image": "russmckendrick/cluster",
    "ImageID": "sha256:53b0bf5f7c7cd98bb2e1d259cb93d222a7612002b761aed511fc6978
↳a49335c2",
    "Labels": {},
    "Mounts": [],
    "Names": [
      "/relaxed_cori"
    ],
    "NetworkSettings": {
      "Networks": {
        "bridge": {
          "Aliases": null,
          "EndpointID": "2707504775f366e46995017d74feacc748fd42e31f8d96bb38dc
↳00d9bc68535f",
          "Gateway": "172.17.0.1",
          "GlobalIPv6Address": "",
          "GlobalIPv6PrefixLen": 0,
          "IPAMConfig": null,
          "IPAddress": "172.17.0.2",
          "IPPrefixLen": 16,
          "IPv6Gateway": "",
          "Links": null,
          "MacAddress": "02:42:ac:11:00:02",
          "NetworkID": "a42aaaf143a718fec2d790615dce9edcd03b84ed3176c270193e
↳0fcb4ff945c4"
        }
      }
    },
    "Ports": [
      {
        "IP": "0.0.0.0",
        "PrivatePort": 80,
        "PublicPort": 32794,
        "Type": "tcp"
      }
    ]
  }
]
```

```
        ],
        "State": "running",
        "Status": "Up 34 seconds"
    }
]
```

Jak widzisz, w zasadzie są to te same dane, które klient Dockera zwraca po uruchomieniu polecenia:

```
$ docker container ls
```

Bardziej szczegółowe informacje na temat wybranego aktywnego kontenera możesz uzyskać, uruchamiając następujące polecenie (pamiętaj o konieczności użycia identyfikatora własnego kontenera):

```
$ curl --unix-socket /var/run/docker.sock
"http://v1.27/containers/a4c2c76ace78e25e1c42833967447042f8607c453c14f443b73
0f605f2e65039/json" | python -m json.tool
```

Polecenie to zwraca zbyt wiele informacji, aby pokazać je tutaj (dane te zajmują około 200 linii). Zwrócone informacje są bardziej rozwlekłą formą informacji wyświetlanych po uruchomieniu polecenia:

```
$ docker container inspect nazwa_kontenera
```

Interfejs programistyczny może być używany w celu odkrywania usług, ale nie polecam używania go w tym celu, ponieważ nauka jego obsługi jest czasochłonna i istnieją lepsze narzędzia przeznaczone do wykonywania tego zadania, takie jak Consul i etcd. Więcej informacji na temat interfejsu programistycznego Dockera, w tym także jego pełną definicję, znajdziesz na stronie <https://docs.docker.com/engine/api/v1.29/>.

Projekt Moby Project

Podczas konferencji DockerCon 2017 ogłoszono założenia projektu **Moby Project**. Wzbudziło to zainteresowanie moich kolegów z branży, ponieważ wydawało się im, że autorzy Dockera planują wydanie kolejnego systemu konteneryzacji.

Co im powiedziałem? Po kilku dniowych przemyśleniach przekazałem im następującą informację:

Moby Project to otwarty projekt będący zbiorem kilku bibliotek używanych podczas budowy systemów opartych na kontenerach. Projekt ten obejmuje również platformę programistyczną umożliwiającą łączenie tych bibliotek, co pozwala na uzyskanie użytecznego systemu. Ponadto Moby Project zawiera również system referencyjny o nazwie Moby Origin — jest to przykładowe, proste narzędzie pozwalające na budowanie i personalizowanie własnej wersji platformy Docker.

Osoby, które słyszały tę wypowiedź, zwykle zadawały mi następujące pytanie: co to tak naprawdę oznacza? Odpowiadałem im w następujący sposób:

Moby Project to otwarta „piaskownica” pracowników firmy Docker, z której mogą korzystać wszyscy, którzy chcą dołożyć coś od siebie do tego projektu — rozwijać nowe, a także istniejące funkcje, biblioteki i platformy wchodzące w skład systemu konteneryzacji. W ten sposób rozwijany jest nowatorski system o nazwie Moby Origin, a także właściwy produkt — Docker, który występuje w darmowej otwartej wersji, a także w płatnej wersji ze wsparciem technicznym.

Jeżeli ktoś mnie zapyta o przykład podobnego projektu — projektu, którego charakterystycznymi cechami jest wprowadzenie najnowszego rozwiązania, stabilność otwartego kodu i komercyjne wsparcie — to takiej osobie przytaczam przykład systemu Red Hat Enterprise Linux firmy Red Hat.

Rozwiązanie to można porównać z filozofią przyjętą przez firmę Red Hat wobec systemu Red Hat Enterprise Linux. Środowisko Fedora jest „piaskownicą” przeznaczoną dla programistów systemu operacyjnego firmy Red Hat, która umożliwia wprowadzanie nowych pakietów i funkcji, a także usuwanie przestarzałych komponentów. Nowe funkcje są wprowadzane w środowisku Fedora zwykle rok lub dwa przed pojawieniem się w Red Hat Enterprise Linux (stabilnej, wspieranej, komercyjnej wersji systemu, która jest oparta na rozwiązańach zaczerpniętych z projektu Fedora). Poza tym istnieje jeszcze wersja wspierana przez środowisko entuzjastów systemu Linux — CentOS.

Dlaczego napisałem o tym rozwiązaniu dopiero na końcu książki? W trakcie tworzenia tej książki projekt Moby Project jest dopiero we wstępnej fazie rozwoju. Tak naprawdę to nie wszystkie wymagane komponenty głównych projektów Dockera zostały przeniesione do projektu Moby Project.

Jedynym naprawdę przydatnym komponentem tego projektu jest rama projektowa **LinuxKit**, która spina ze sobą wszystkie biblioteki i pozwala na uzyskanie systemu zdolnego do uruchamiania kontenerów.

Projekt ten jest rozwijany bardzo szybko, a więc nie przedstawię tutaj żadnych zastosowań ramy LinuxKit i nie będę opisywał projektu Moby Project, ponieważ rozwiązania te z pewnością zostałyby zmodyfikowane, zanim ta książka dotarłaby do czytelnika. W zamian poleczę Ci kilka stron, na których są publikowane informacje na temat projektu Moby Project:

- główna strona projektu: <https://mobyproject.org/>;
- strona projektu w repozytorium GitHub: <https://github.com/moby/>;
- konto projektu w serwisie społecznościowym Twitter (dobre źródło informacji o rozwoju projektu i wskazówek dotyczących jego obsługi): <https://twitter.com/moby/>;
- strona główna ramy projektowej LinuxKit zawierająca instrukcje i przykłady ułatwiające rozpoczęcie korzystania z niej: <https://github.com/linuxkit/>.

Własny wkład w rozwój Dockera

Chcesz mieć własny wkład w rozwój platformy Docker? Masz wspaniały pomysł na usprawnienie pracy tej platformy lub jakiegoś jej komponentu? W tym podrozdziale dowiesz się, jak dołączyć do entuzjastów rozwijających Dockera. Nie musisz być programistą — rozwój platformy może być wspierany na wiele sposobów. Z Dockera korzysta ogromna rzesza użytkowników. Możesz po prostu pomagać w rozwiązywaniu napotykanych przez nich problemów. Za chwilę dowiesz się, jak to zrobić.

Rozwój kodu

Największym wkładem, jaki można wnieść w rozwój Dockera, jest pomoc w tworzeniu kodu. Docker jest platformą o charakterze otwartym, a więc możesz pobrać jego kod i spróbować stworzyć nowe funkcje, a następnie wysłać zmodyfikowany kod w celu przeanalizowania go przez innych twórców i zatwierdzenia wprowadzonych przez Ciebie zmian. Już wskutek samego zaakceptowania zaimplementowanych modyfikacji poczujesz dumę.

Możesz rozwijać praktycznie wszystkie elementy platformy Docker (<https://github.com/docker/>) i projektu Moby Project. Jak to zrobić? Najlepiej zacznij od lektury wytycznych znajdujących się w oficjalnej dokumentacji Dockera (<https://docs.docker.com/opensource/ways/>).

Jak zapewne się domyślasz, uruchomienie środowiska programistycznego jest dość proste — większość pracy wykonuje się w kontenerach. Musisz po prostu założyć konto w repozytorium GitHub i zainstalować te trzy narzędzia:

- **Git:** <https://git-scm.com/>;
- **Make:** <https://www.gnu.org/software/make/>;
- **Docker** — jeżeli dotarłeś do tej części tej książki, to powinieneś już dysponować zainstalowaną platformą Docker.

Więcej informacji na temat przygotowywania środowiska programistycznego Dockera w systemach macOS i Linux znajdziesz na stronie <https://docs.docker.com/docker-for-mac/>. Jeżeli korzystasz z systemu Windows, to zajrzyj na stronę <https://docs.docker.com/docker-for-windows/>.

Twórcy każdego dobrego otwartego projektu muszą stosować się do pewnych wytycznych. Polecam Ci lekturę wspaniałego poradnika dla początkujących znajdującego się na stronie <https://docs.docker.com/get-started/>. Zapoznaj się również z bardziej szczegółową dokumentacją przepływu pracy znajdującą się na stronie <https://docs.docker.com/opensource/>.

Twórcy Dockera pracują nad przyjęciem formalnego kodeksu rozwoju platformy. Do momentu wprowadzenia go w życie proszą o stosowanie się do postanowień następującej przysięgi:

W celu wspierania otwartego i przyjaznego środowiska my — jako współpracownicy i opiekunowie projektu — zobowiązujemy się dopilnować, aby uczestnictwo w naszym projekcie oraz przynależność do jego społeczności nie były narażone na nękanie, bez względu na budowę ciała, doświadczenie, kolor skóry, narodowość, niepełnosprawność, orientację i identyfikację seksualną, pochodzenie etniczne, religię, wiek oraz wygląd zewnętrzny.

Wśród przykładów zachowania, którego nie będziemy akceptować, są:

- Używanie języka i grafik o podtekście seksualnym, okazywanie niepożądanego zainteresowania seksualnego, a także zaloty.
- Trollowanie, obraźliwe bądź urągające komentarze oraz ataki osobiste bądź polityczne.
- Nękanie, zarówno na forum publicznym, jak i prywatnie.
- Publikowanie danych osobistych innych osób — takich jak adres fizyczny czy elektroniczny — bez ich wyraźnej zgody.
- Inne zachowania, które mogłyby zostać uznane za nieodpowiednie w kontekście profesjonalnym. Przypadki nadużyć, nękania i innych nieakceptowalnych zachowań.

Przypadki nękania, gróżb oraz innych form nieakceptowalnego zachowania mogą być zgłoszane do zespołu projektu.

Zasady te stanowią fragment konwencji dobrego zachowania opublikowanych na stronie <https://www.contributor-covenant.org/pl/version/1/4/code-of-conduct.html>.

Pomoc innym

W rozwoju platformy Docker możesz pomagać, nie tylko tworząc kod i rozwijając nowe funkcje. Możesz wesprzeć innych swoją wiedzą. Środowisko użytkowników Dockera jest bardzo otwarte i wiele osób oferuje swoją pomoc innym. Sam korzystałem ze wsparcia innych, rozwiązuje skomplikowane problemy. Poza korzystaniem z pomocy innych warto odwdzięczyć się społeczności i zaoferować w zamian coś od siebie. Fora dyskusyjne użytkowników Dockera warto przeglądać w celu podejrzenia innych rozwiązań. Ponadto, analizując konfigurację środowisk i napotykane problemy, możesz lepiej przemyśleć proces tworzenia własnego środowiska.

Warto również śledzić wątki dyskusyjne dotyczące usług w repozytorium GitHub. Znajdziesz tam prośby o dodanie funkcjonalności, a także pomysły implementacji różnych rozwiązań opracowane w wyniku użytkowania różnych usług. Możesz pomóc w sprawdzaniu problemów napotkanych przez innych — poprzez odtworzenie problematycznej sytuacji możesz znaleźć rozwiązanie czyegoś problemu.

Społeczność wspierająca Dockera jest bardzo aktywna. Znajdziesz ją na stronie <https://community.docker.com>. Na stronie tej publikowane są wiadomości związane ze społeczeństwem, a także informacje o ważnych wydarzeniach. Ponadto serwis ten pozwala na skontaktowanie się za pomocą kanałów Slack z innymi użytkownikami Dockera i programistami. W momencie pisania tej książki istniało ponad 80 kanałów dotyczących różnych tematów, między innymi Dockera dla systemu macOS, Dockera dla systemu Windows, systemu Alpine Linux, technologii Swarm, magazynów danych i sieci. Cały czas na stronie czeka na Ciebie kilkuset aktywnych użytkowników.

Warto również korzystać z forum Dockera znajdującego się na stronie <https://forums.docker.com/>. Jest to dobre źródło wiedzy, które możesz przeszukiwać za pomocą słów kluczowych i konkretnych tematów (problemów).

Inny wkład w rozwój Dockera

W rozwoju Dockera możesz pomagać również na inne sposoby. Możesz promować tę platformę i wzbudzać zainteresowanie nią w firmie, w której pracujesz — rozpocząć dyskusję na jej temat na spotkaniu, na grupie dyskusyjnej, liście mailingowej lub zaplanować cykliczne spotkania dyskusyjne. Takie spotkania nie muszą się ograniczać do jednej firmy — można je skierować do ogółu osób związanych z IT mieszkających w okolicy, co poprawi kontakty pomiędzy osobami pracującymi w różnych firmach. Informacje o miejscowościach, w których odbywają się takie spotkania, można znaleźć między innymi na stronie <https://events.docker.com/>.

Podsumowanie

W tym rozdziale opisaliśmy zagadnienia związane z narzędziami przeznaczonymi do odkrywania usług, które mogą rozszerzyć możliwości Twojej aplikacji.

Co prawda zwykle nie musisz korzystać z aż tak wysokiego stopnia integracji z platformą hosta, ale w celu zminimalizowania potencjalnych problemów warto połączyć odkrywanie usług z technologiami opisanymi w poprzednich rozdziałach, takimi jak np. Docker Swarm. Zabieg taki pozwoli na zapewnienie wdrażania aplikacji w wysoce dostępnej i automatycznie poprawiającej się platformie, która może pełnić funkcję środowiska produkcyjnego.

Na koniec skrótnie opisaliśmy, jak możesz wnieść wkład w rozwój platformy Docker i projektu Moby Project poprzez tworzenie kodu, pomoc innym użytkownikom Dockera lub po prostu poprzez informowanie o możliwościach oferowanych przez technikę konteneryzacji.

Skorowidz

A

ADD, 38
administratorzy, 19
Amazon
 EC2 Container Registry, 78
 Elastic Block Storage, 290
 Elastic Compute Cloud, 290
 Elastic Container Service, 290
 Web Services, 121, 227
analiza pliku Dockerfile, 37
API, 304
aplikacja
 do głosowania, 139
 Docker Bench Security, 252
 Mobycounter, 133, 137
 NGINX, 50
architektury referencyjne, 293
attach, 88
Auto Scaling, 291
automatyzacja budowy obrazu, 63
AWS, Amazon Web Services, 121, 227

B

bezpieczeństwo, 262
 kontenerów, 245
 platformy Docker, 245
budowane
 pliki kontenerów, 252
 obrazów, 42, 48
 automatyzacja, 63
 Dockera, 42
 własnych, 43
build, 148

chmura
 DigitalOcean, 119, 128
 Docker Cloud, 223
 Google Cloud Storage, 74
 Microsoft Azure, 74
 VPC, 122
chmury
 prywatne, 122
 uruchamianie hostów, 119
Clair, 263
CMD, 40
config, 148
Consul, 295
dostępność, 304
szablony narzędzia, 302
uruchamianie narzędzia, 296
usługa Registrar, 296
zapytania, 297
COPY, 38
CoreOS, 128
create, 148

D

degradacja węzła menedżera, 165
demon Dockera, 257
diff, 249
DigitalOcean, 119, 128, 226
dobre praktyki, 250
Docker, 17
 Bench Security, 252
 analiza informacji, 255
 konfiguracja demona, 256
 konfiguracja hosta, 255

- Docker
obrazy kontenerów, 257
operacje bezpieczeństwa, 260
pliki konfiguracyjne demona, 257
środowisko robocze kontenerów, 258
uruchamianie narzędzi, 253
Cloud, 33, 223
Amazon Web Services, 239
łączenie kont, 225
tryb Swarm, 238
uruchamianie stosu, 234
uruchamianie węzłów, 231
zakładanie konta, 224
Community Edition, 32
Compose, 33, 133
polecenia, 146
uruchamianie aplikacji, 135
dla Azure, 33
dla systemu macOS, 33
dla systemu Windows, 33
dla usług Amazon Web Services, 33
Engine, 32, 246
Enterprise Edition, 32
Hub, 33, 57
konfiguracja serwisu, 64
menu Create, 60
panel główny, 58
profil, 61
przycisk Explore, 59
przycisk Organizations, 60
strona Stars, 62
ustawienia, 61
Machine, 33, 113
uruchamianie hostów, 119
wdrażanie lokalnych hostów, 114
Registry, 71
wdrażanie rejestru, 72
Store, 33, 70
Swarm, 33, 155
degradacja węzła menedżera, 165
dołączanie wykonawców, 160
drenaż węzła, 166
instalacja, 156
kasowanie klastra, 173
listy węzłów, 161
menedżer Swarm, 157
nakładki sieciowe, 175
promowanie hosta, 164
role, 156
równoważenie obciążeń, 174
stosy, 168, 171
tworzenie harmonogramu, 176
tworzenie klastra, 158
usługi, 168
wykonawca Swarm, 157
zarządzanie klastrem, 161
Trusted Registry, 74, 293
dołączanie wykonawców, 160
down, 153
drenaż węzła, 166
dystrybucja obrazów, 57
dzienniki, 91, 186
- E**
- EBS, 290
EC2, 290
ekosystem, 32
ekran główny, 58
Elastic Load Balancing, 290
ELB, 290
ENTRYPOINT, 40
ENV, 41
etcd, 304
events, 149
exec, 89, 151
EXPOSE, 40
- F**
- FROM, 37
- G**
- GitHub, 76
Grafana, 280, 282
- H**
- harmonogram, 176
Heroku, 289
hosty
 dedykowane, 22
 Dockera, 246, 291
 lokalne, 114
 robocze, 164
 uruchamianie, 119

I

- informacje
 - o obciążeniu procesora, 186
 - o pamięci, 186
 - o połączeniach sieciowych, 186
 - o procesach, 91
- instalacja, 23
 - Docker Swarm, 156
 - Rancher, 202
 - uwierzytelniania, 201
 - w systemie Linux, 24
 - w systemie macOS, 25
 - w systemie Windows 10, 27
- instalator Tectonic, 286
- interfejs Docker API, 304

K

- kasowanie klastra Swarm, 173
- kill, 96, 153
- klaster, 161, 218, 292
 - Swarm, 173, 195
 - tworzenie, 192
- klient Dockera, 29
- komunikacja klastra, 293
- konfiguracja
 - demona Dockera, 251, 256
 - hosta, 251, 255
 - serwisu Docker Hub, 64
 - uwierzytelniania, 201, 204
- kontenery, 35, 44, 183
 - bezpieczeństwo, 245
 - komunikacja, 88
 - podgląd właściwości, 216
 - polecenia, 84, 95
 - stany, 95
 - usuwanie, 97
 - zarządzanie, 83

L

- LABEL, 37
- Linux Containers, 22
- listy węzłów, 161
- logs, 91, 149

Ł

- ładowanie własnych obrazów, 68

M

- maszyna wirtualna, 22
 - VirtualBox, 130
- menedżer Swarm, 157
- menu Create, 60
- Microbadger, 79
- mieszanie procesów, 291
- Moby Project, 306
- Mobycounter, 137
- monitorowanie, 277
 - narzędzia, 285

N

- nakładki sieciowe, 175
- narzędzia do monitorowania, 285
- narzędzie
 - cadvisor, 279
 - Consul, 296
 - Docker Bench Security, 252
 - Docker Machine, 113
 - etcd, 304
 - Grafana, 280, 282
 - Microbadger, 79
 - node-exporter, 279
 - Portainer, 177
 - Rancher, 201
 - WP-CLI, 274

O

- obciążenia wejściowe, 174
- obrazy, 187, 252
 - Dockera, 42
 - dystrybucja, 57
 - kontenerów, 35, 257
 - przechowywanie, 57
 - rejestry, 293
 - zaufane źródła, 247
- obsługa
 - klastrów, 292
 - kontenerów, 84
 - kontenerów LXC, 22
 - sieci, 100
 - volumenów, 100

ograniczenia zasobów, 93
ONBUILD, 41
operacje bezpieczeństwa, 252, 260
organizacja Center for Internet Security, 251

P

panel główny, 58
pause, 96, 149
platforma Heroku, 289
plik
 .dockerignore, 43
 Dockerfile, 35
 analiza, 37
 budowa obrazu, 42
 dobre praktyki, 41
 polecenia, 37, 40
 zmienne środowiskowe, 50
 YAML, 137
pliki konfiguracyjne demona Dockera, 251, 257
podgląd właściwości, 216
polecenia
 Docker Compose, 146
 Dockera, 247
 obsługa kontenerów, 84
 pliku Dockerfile, 40
polecenie
 ADD, 38
 attach, 88
 build, 148
 CMD, 40
 config, 148
 COPY, 38
 create, 148
 diff, 249
 docker build, 42
 down, 153
 ENTRYPOINT, 40
 ENV, 41
 events, 149
 exec, 89, 151
 EXPOSE, 40
 FROM, 37
 kill, 96, 153
 LABEL, 37
 logs, 91, 149
 ONBUILD, 41
 pause, 96, 149
 prune, 97

ps, 147
pull, 148
restart, 96
restart, 149
rm, 98, 153
run, 38, 151, 247
scale, 152
service, 168
start, 96, 149
stats, 93
stop, 96, 149
top, 92, 149
unpause, 96, 149
up, 147
USER, 41
WORKDIR, 41
Portainer, 177
Docker, 191
dzienniki, 186
konsola, 187
kontenery, 183
obrazy, 187
panel główny, 181
sieci, 190
statystyki, 186
stosowanie, 180
szablon aplikacji, 181
uruchamianie, 178
usługi, 193, 196
węzły końcowe, 198
wolumeny, 190
zdarzenia, 190
prace programistyczne, 265
profil, 61
projekt Moby Project, 306
przechowywanie obrazów, 57
przepływ zadań, 265
przycisk
 Explore, 59
 Organizations, 60
PS, 147
pull, 148

Q

Quay, 75, 262

R

Rancher, 201
 instalacja, 202
 katalog, 217
 konfiguracja uwierzytelniania, 204
 tworzenie stada, 207
 uruchamianie stosów, 209
 usuwanie stada, 217
 RancherOS, 130
 Registrator, 296
 rejestr
 Amazon EC2 Container Registry, 78
 Docker Registry, 71
 Docker Trusted Registry, 74
 rejestry
 niezależne, 75
 obrazów, 293
 repozytorium, 63
 Docker Hub, 57
 GitHub, 76
 restart, 96
 restart, 149
 rm, 153
 role Docker Swarm, 156
 rozszerzenie Weave Net, 124
 run, 38, 151, 247

S

scale, 152
 serwis
 Docker, 58
 Docker Hub, 57
 Docker Store, 70
 sieci, 100, 190
 silnik Consul, 295
 skanowanie zabezpieczeń Dockera, 260
 stany kontenerów, 95
 start, 96, 149
 stats, 93
 stop, 96, 149
 stos, 171, 209, 210, 234
 stosy Docker Swarm, 168
 strona Stars, 62
 Swift, 74
 system
 CoreOS, 128
 Kubernetes, 286

plików, 74
 RancherOS, 130
 szablony
 aplikacji, 181
 narzędzia Consul, 302

Ś

środowisko
 produkcyjne, 291
 robocze kontenerów, 252, 258
 wykonawcze, 252

T

Tectonic, 286
 top, 92, 149
 tryb Swarm, 238
 tworzenie
 harmonogramu, 176
 klastra, 158, 192
 obrazów kontenerów, 35
 stada, 207

U

unpause, 96, 149
 up, 147
 uruchamianie
 aplikacji, 135
 hostów, 119
 narzędzia Consul, 296
 stosów, 209, 234
 usługi Registrator, 296
 węzłów, 231
 USER, 41
 usługa, 196
 Amazon Elastic Container Service, 290
 Amazon S3, 74
 Docker Cloud, 223
 Docker Swarm, 168
 Portainer, 193
 Quay, 75
 Registrator, 296
 ustawienia, 61
 usuwanie
 kontenerów, 97
 stada, 217
 uwierzytelnianie, 201, 204

V

VPC, Virtual Private Cloud, 122, 291

W

wdrażanie

 lokalnych hostów, 114
 rejestru, 72

węzły

 końcowe, 198
 menedżera, 165
 uruchamianie, 231

wiersz poleceń, 29

VirtualBox, 130

właściwości kontenerów, 216

wolumeny, 107, 190

WORKDIR, 41

wykonawca Swarm, 157

wykrywanie usług, 295

Z

zabezpieczenia Dockera, 260

zarządzanie
 klastrem, 161
 kontenerami, 83

zasoby, 93

zaufane źródła obrazów, 247

zdarzenia, 190

zewnętrzne platformy, 286

zmienne środowiskowe, 50

ż

źródła obrazów, 247

ż

żądania do kontenerów, 292

żądanie SIGTERM, 96

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ 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 WYDAWNICZA

 Helion SA