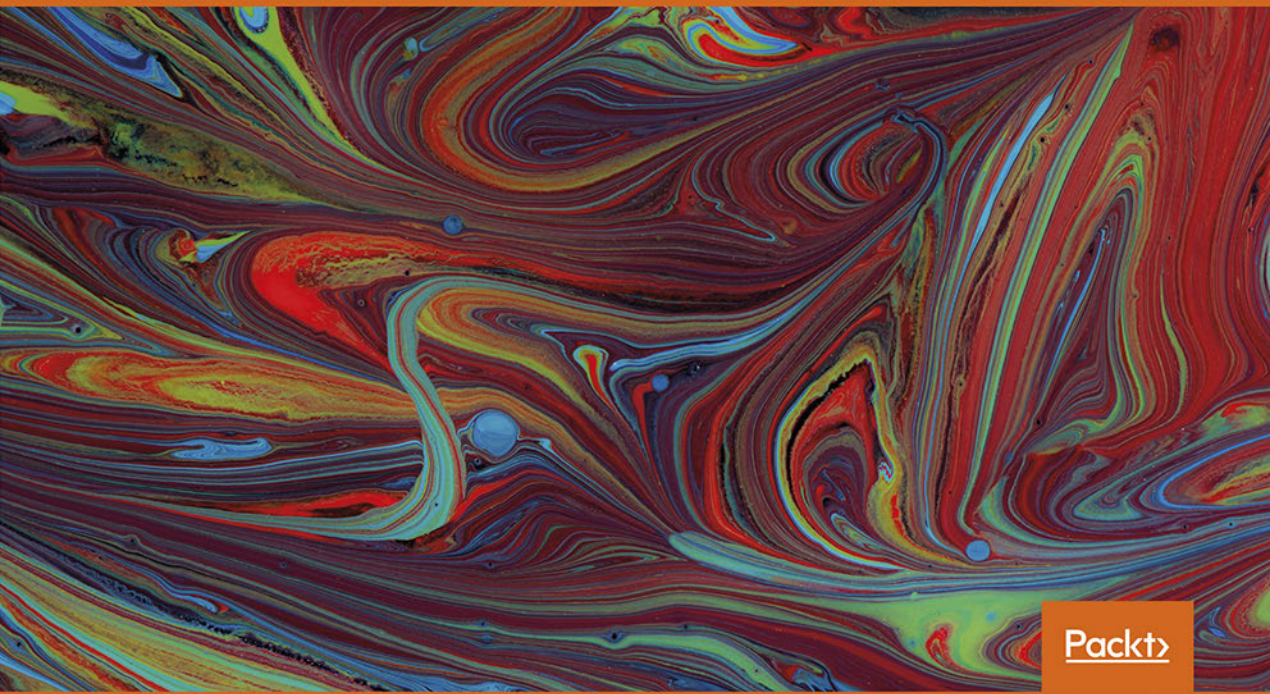


Docker dla programistów

Rozwijanie aplikacji
i narzędzia ciągłego dostarczania DevOps



Packt 

Richard Bullington-McGuire,
Michael Schwartz, Andrew K. Dennis

Tytuł oryginału: Docker for Developers: Develop and run your application with Docker containers using DevOps tools for continuous deliver

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-7727-1

Copyright © Packt Publishing 2020. First published in the English language under the title 'Docker for Developers – (9781789536058)'.

Polish edition copyright © 2021 by Helion S.A.

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 jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

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

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

Helion S.A.

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/dockpr>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

O autorach	15
O recenzencie	17
Przedmowa	19
Część I. Wprowadzenie do platformy Docker — kontenery i programowanie w lokalnym środowisku	25
Rozdział 1. Wprowadzenie do platformy Docker	27
Geneza platformy Docker	28
Hosting kolokacyjny	28
Hosting samodzielny	29
Centra danych	29
Wirtualizacja jako sposób na ekonomiczne wykorzystanie zasobów	31
Rosnące zapotrzebowanie na energię	33
Wirtualizacja i chmura obliczeniowa	34
Dalsza optymalizacja zasobów centrów danych przy użyciu kontenerów	36
Podsumowanie	38
Dalsza lektura	38
Rozdział 2. Tworzenie aplikacji z użyciem VirtualBox i kontenerów Docker	39
Wymagania techniczne	40
Problem zanieczyszczenia systemu plików hosta	40
Tworzenie maszyn wirtualnych za pomocą programu VirtualBox	41
Wprowadzenie do wirtualizacji	41
Tworzenie maszyny wirtualnej	42
Dodatki do systemu gościa	43
Instalacja programu VirtualBox	44

Kontenery Docker	45
Wprowadzenie do kontenerów	45
Tworzenie aplikacji przy użyciu platformy Docker	47
Pierwsze kroki z platformą Docker	48
Automatyzacja poleceń za pomocą skryptów	49
Podsumowanie	59
Dalsza lektura	60
Rozdział 3. Udostępnianie kontenerów w serwisie Docker Hub	61
Wymagania techniczne	62
Wprowadzenie do serwisu Docker Hub	62
Korzystanie z serwisu Docker Hub za pomocą wiersza poleceń	63
Korzystanie z serwisu Docker Hub za pomocą przeglądarki	64
Implementacja kontenera MongoDB w aplikacji	66
Uruchomienie powłoki kontenera	69
Wprowadzenie do architektury mikrousługowej	71
Skalowalność	72
Komunikacja między kontenerami	72
Implementacja prostej aplikacji mikrousługowej	75
Udostępnianie kontenerów w serwisie Docker Hub	79
Podsumowanie	82
Dalsza lektura	82
Rozdział 4. Tworzenie systemów przy użyciu kontenerów	83
Wymagania techniczne	84
Wprowadzenie do narzędzia Docker Compose	84
Problem ze skryptami	85
Pliki konfiguracyjne narzędzia Docker Compose	86
Dziedziczenie konfiguracji	89
Sekcja depends_on	90
Definiowanie udostępnianych portów	91
Lokalne sieci w platformie Docker	94
Definiowanie sieci za pomocą skryptów	94
Tworzenie sieci za pomocą narzędzia Docker Compose	96
Wiązanie systemów plików hosta i kontenera	97
Optymalizacja wielkości kontenera	98
Skrypt build.sh	100
Inne narzędzia kompozycyjne	101
Docker Swarm	101
Kubernetes	101
Packer	102
Podsumowanie	102
Dalsza lektura	103

Część II. Platforma Docker w środowisku produkcyjnym 105

Rozdział 5. Wdrażanie i uruchamianie kontenerów w środowisku produkcyjnym 107

Wymagania techniczne	108
Przykładowa aplikacja Shipt Clicker	108
Uruchamianie kontenerów Docker w środowisku produkcyjnym	109
Minimalne środowisko produkcyjne	109
Niezbędne minimum — Docker i Docker Compose na jednym hoście	110
Wsparcie dla platformy Docker	110
Problemy z wdrażaniem na pojedynczym hoście	110
Zarządzane usługi chmurowe	111
Google Kubernetes Engine	111
AWS Elastic Beanstalk	112
AWS ECS i Fargate	112
AWS EKS	112
Microsoft Azure Kubernetes Service	113
DigitalOcean Docker Swarm	113
Tworzenie własnych klastrów Kubernetes	113
Dobieranie właściwej konfiguracji produkcyjnej	114
Ćwiczenie — dołącz do zespołu Shipt Clicker	116
Ćwiczenie — wybór właściwej metody wdrożenia	119
Ćwiczenie — ocena plików Dockerfile i docker-compose.yml	121
Podsumowanie	121

Rozdział 6. Wdrażanie aplikacji przy użyciu Docker Compose 123

Wymagania techniczne	124
Przykładowa aplikacja — Shipt Clicker v2	124
Dobór sprzętu i systemu operacyjnego dla aplikacji jednoserwerowej	124
Wymagania dla wdrożenia jednoserwerowego	124
Przygotowanie hosta do uruchomienia platformy Docker i narzędzia Docker Compose	125
Instalacja platformy Docker i narzędzia Git	126
Wdrażanie aplikacji przy użyciu plików konfiguracyjnych i skryptów	127
Weryfikacja pliku Dockerfile	127
Weryfikacja pliku docker-compose.yml	129
Przygotowanie produkcyjnego pliku .env	131
Skrypty	132
Ćwiczenie — przechowywanie plików aplikacyjnych poza serwerem produkcyjnym	135
Ćwiczenie — zabezpieczenie środowiska produkcyjnego	135
Monitorowanie niewielkich aplikacji — dzienniki i alarmy	136
Dzienniki	136
Alarmy	137
Ograniczenia aplikacji jednoserwerowych	137
Brak automatycznego przełączania awaryjnego	138
Brak skalowalności w poziomie wraz ze wzrostem obciążenia	138
Niestabilność działania z powodu błędnej konfiguracji	138
Katastrofalne skutki awarii w przypadku braku kopii zapasowej	139

Studium przypadku — migracja z systemu CoreOS i usługi DigitalOcean do CentOS 7 i AWS	139
Podsumowanie	139
Dalsza lektura	140
Rozdział 7. Ciągłe wdrażanie oprogramowania przy użyciu systemu Jenkins	141
Wymagania techniczne	142
Przykładowa aplikacja — Shiplt Clicker v3	142
Wykorzystanie systemu Jenkins w procesie ciągłej integracji oprogramowania	143
Pułapki, których powinieneś unikać	143
Wykorzystanie istniejącego serwera	144
Instalacja systemu Jenkins	144
Ciągłe wdrażanie oprogramowania przy użyciu systemu Jenkins	148
Plik Jenkinsfile i połączenie z serwerem	148
Testowanie systemu Jenkins i platformy Docker za pomocą skryptu procesowego	148
Łączenie z serwerem produkcyjnym za pomocą usługi SSH	150
Modyfikowanie konfiguracji za pomocą systemu Jenkins	154
Umieszczenie pliku Jenkinsfile w serwisie GitHub	154
Zmiana źródeł wszystkich repozytoriów	156
Definiowanie zmiennych środowiskowych dla serwera produkcyjnego	157
Budowanie kontenerów i umieszczanie ich w serwisie Docker Hub	158
Umieszczanie kontenerów w serwisie Docker Hub i wdrażanie ich w środowisku produkcyjnym	159
Wdrażanie różnych odgałęzień oprogramowania w kilku środowiskach	162
Utworzenie środowiska testowego	162
Definiowanie zmiennych środowiskowych dla serwera testowego	163
Wymuszenie wdrożenia testowego odgałęzienia projektu	163
Złożoność i ograniczenia skalowalności systemu Jenkins	165
Zarządzanie wieloma hostami	165
Złożoność skryptów	166
Kiedy wiadomo, że została osiągnięta granica?	166
Podsumowanie	167
Dalsza lektura	167
Rozdział 8. Wdrażanie kontenerów Docker przy użyciu platformy Kubernetes	169
Wymagania techniczne	170
Opcje lokalnej instalacji platformy Kubernetes	170
Docker Desktop i platforma Kubernetes	170
Minikube	171
Sprawdzenie poprawności działania platformy Kubernetes	172
Wdrożenie przykładowej aplikacji Shiplt Clicker v4	172
Instalacja programu Helm	172
Lokalne wdrożenie aplikacji Shiplt Clicker i kontrolera NGINX Ingress Controller	173
Dobór dystrybucji platformy Kubernetes	175
Google Kubernetes Engine	175
AWS EKS	175
Red Hat OpenShift	176
Microsoft Azure Kubernetes Service	176
Inne opcje	177

Pojęcia stosowane w platformie Kubernetes	178
Obiekty	178
Mapy ConfigMap	179
Pody	180
Węzły	180
Usługi	180
Kontrolery wejściowe	181
Skrytki	182
Przestrzenie nazw	187
Konfigurowanie usługi AWS EKS za pomocą szablonu CloudFormation	187
Wprowadzenie do szablonów AWS EKS Quick Start CloudFormation	188
Przygotowanie konta AWS	188
Szablony AWS EKS Quick Start CloudFormation	192
Konfigurowanie klastra EKS	195
Wdrożenie aplikacji w klastrze AWS EKS i ograniczenie zasobów	197
Konfigurowanie ograniczeń chroniących przed wyciekami pamięci i przeciążeniem procesora	197
Przygotowanie aplikacji Shipt Clicker do korzystania z kontrolera ALB	198
Wdrożenie aplikacji Shipt Clicker w klastrze EKS	198
Repozytorium AWS Elastic Container Registry w klastrze AWS EKS	199
Tworzenie repozytorium ECR	200
Rozdzielanie środowisk za pomocą etykiet i przestrzeni nazw	202
Przykład — oznaczenie etykietami środowisk w domyślnej przestrzeni nazw	202
Środowiska programistyczne, akceptacyjne, testowe i produkcyjne	203
Podsumowanie	204
Dalsza lektura	204
Rozdział 9. Ciągłe wdrażanie oprogramowania w chmurze przy użyciu platformy Spinnaker	207
<hr/>	
Wymagania techniczne	208
Zaktualizowana wersja aplikacji Shipt Clicker v5	208
Usprawnienie platformy Kubernetes pod kątem utrzymywania aplikacji	209
Zarządzanie klastrem EKS za pomocą lokalnej stacji roboczej	209
Diagnostowanie problemów z połączeniem narzędzia kubectl z klastrem	210
Przełączanie pomiędzy kontekstem klastra i lokalnej stacji	210
Sprawdzenie poprawności działania kontrolera wejściowego ALB	211
Przygotowanie domeny Route 53 i certyfikatu	211
Utworzenie i wdrożenie aplikacji Shipt Clicker v5	212
Platforma Spinnaker — kiedy i dlaczego są niezbędne bardziej zaawansowane wdrożenia	215
Wprowadzenie do platformy Spinnaker	215
Instalacja platformy Spinnaker w klastrze AWS EKS za pomocą programu Helm	217
Komunikacja z platformą Spinnaker za pomocą proxy kubectl	218
Udostępnianie platformy za pomocą kontrolera wejściowego ALB	218
Konfiguracja platformy Spinnaker za pomocą programu Halyard	220
Połączenie platformy Spinnaker z systemem Jenkins	220
Integracja systemu Jenkins z platformą Spinnaker i repozytorium ECR	221
Połączenie platformy Spinnaker z serwisem GitHub	226

Połączenie platformy Spinnaker z serwisem Docker Hub	226
Diagnozowanie problemów z platformą Spinnaker	227
Prosta strategia wdrożenia aplikacji Shiplt Clicker za pomocą platformy Spinnaker	228
Definiowanie aplikacji w platformie Spinnaker	228
Definiowanie procesu w platformie Spinnaker	229
Utworzenie wpisu DNS dla kontrolera wejściowego	234
Funkcjonalności wdrożeniowe i testowe platformy Spinnaker	235
Wdrożenie kanarkowe	235
Wdrożenie „czerwone/czarne”	235
Anulowanie wdrożenia	236
Testowanie aplikacji	236
Podsumowanie	237
Dalsza lektura	237
Rozdział 10. Monitorowanie kontenerów Docker przy użyciu systemów Prometheus, Grafana i Jaeger	239
<hr/>	
Wymagania techniczne	240
Wdrożenie demonstracyjnej aplikacji Shiplt Clicker v7	240
Dzienniki kontenerów Docker i programów uruchomieniowych	243
Dzienniki kontenerów Docker	243
Cechy idealnego systemu zarządzania dziennikami	244
Diagnozowanie problemów z warstwą sterowania platformy Kubernetes na podstawie dzienników	245
Zapisywanie dzienników w usłudze CloudWatch Logs	246
Długotrwałe przechowywanie dzienników w usłudze S3	247
Analiza dzienników za pomocą usług CloudWatch Insights i Amazon Athena	248
Ćwiczenie — sprawdzenie liczby uruchomień gry Shiplt Clicker	249
Testy dostępności, gotowości i uruchamiania w platformie Kubernetes	249
Sprawdzanie za pomocą testów dostępności, czy kontener odpowiada na zapytania	250
Sprawdzanie za pomocą testów gotowości, czy usługa może przetwarzać ruch	250
Przystosowanie aplikacji Shiplt Clicker do osobnych testów dostępności i gotowości	251
Ćwiczenie — wymuszenie negatywnego wyniku testu gotowości aplikacji Shiplt Clicker	252
Zbieranie wskaźników i wysyłanie alarmów za pomocą systemu Prometheus	252
Historia systemu	253
Zapytania i interfejs WWW	253
Definiowanie w aplikacji wskaźników dla systemu Prometheus	254
Odczytywanie niestandardowych wskaźników	256
Konfiguracja alarmów	256
Wysyłanie powiadomień za pomocą modułu Alertmanager	258
Szczegóły zapytań i zewnętrznego monitoringu	260
Wizualizacja danych operacyjnych za pomocą systemu Grafana	260
Dostęp do systemu	260
Dodawanie paneli opracowanych przez społeczność użytkowników	261
Utworzenie nowego panelu z niestandardowym zapytaniem	262

Monitorowanie wydajności aplikacji za pomocą systemu Jaeger	264
Interfejs OpenTracing API	264
Wprowadzenie do systemu Jaeger	265
Instalacja klienta systemu Jaeger w aplikacji Shiplt Clicker	267
Instalacja rozszerzenia Jaeger Operator	270
Podsumowanie	272
Dalsza lektura	272
<hr/> Rozdział 11. Skalowanie i testy obciążeniowe aplikacji w środowisku Docker	275
Wymagania techniczne	276
Nowa aplikacja Shiplt Clicker v8	276
Skalowanie klastra Kubernetes	278
Ręczne skalowanie klastra	279
Dynamiczne skalowanie klastra	281
Siatka usług Envoy i jej zastosowania	285
Zarządzanie ruchem w sieci	286
Instalacja siatki Envoy	287
Testowanie skalowalności i wydajności aplikacji za pomocą narzędzia k6	291
Rejestrowanie i odtwarzanie sesji	292
Ręczne tworzenie realistycznego testu	293
Wykonanie testu obciążeniowego	297
Podsumowanie	298
Dalsza lektura	299
<hr/> Część III. Bezpieczeństwo kontenerów Docker	301
<hr/> Rozdział 12. Wprowadzenie do bezpieczeństwa kontenerów	303
Wymagania techniczne	304
Wirtualizacja i modele bezpieczeństwa hipervizora	304
Wirtualizacja i pierścienie ochronne	304
Platforma Docker i pierścienie ochronne	306
Kontenerowe modele bezpieczeństwa	308
Docker Engine, containerd i zabezpieczenia w systemie Linux	309
Przestrzeń PID	310
Przestrzeń MNT	311
Przestrzeń NET	311
Przestrzeń IPC	311
Przestrzeń UTS	311
Przestrzeń USER	312
Uwaga dotycząca grup cgroups	312
Dobre praktyki w skrócie	312
Regularnie instaluj poprawki	313
Zabezpieczaj gniazdo sieciowe	313
Nie uruchamiaj błędnego kodu	315
Zawsze twórz konto użytkownika z minimalnymi uprawnieniami	315
Podsumowanie	315

Rozdział 13. Podstawy bezpieczeństwa i dobre praktyki korzystania z platformy Docker	317
Wymagania techniczne	318
Bezpieczeństwo obrazów kontenerów	318
Weryfikacja obrazu	320
Minimalny obraz bazowy	322
Ograniczanie uprawnień	323
Zapobieganie wyciekowi danych	324
Bezpieczne korzystanie z poleceń w platformie Docker	326
Polecenia COPY i ADD — jaka jest różnica?	326
Kopiowanie rekurencyjne — bądź ostrożny	327
Bezpieczeństwo procesu budowania kontenerów	328
Wieloetapowy proces budowania obrazu	329
Ograniczanie możliwości i zasobów wdrażanego kontenera	330
Ograniczanie zasobów	330
Ograniczanie możliwości	331
Podsumowanie	332
Rozdział 14. Zaawansowane zabezpieczenia: skrytki, poufne polecenia, znaczniki i etykiety	333
Wymagania techniczne	334
Bezpieczne przechowywanie poufnych danych w platformie Docker	334
Dziennik Raft	335
Tworzenie, edytowanie i usuwanie skrytek	336
Tworzenie skrytek	336
Odczytywanie skrytek	336
Usuwanie skrytek	337
Skrytki w akcji — przykłady	338
Zabezpieczanie kontenerów za pomocą znaczników	340
Umieszczanie w etykietach metadanych aplikacji	341
Podsumowanie	342
Rozdział 15. Skanowanie, monitorowanie i zewnętrzne narzędzia	343
Wymagania techniczne	344
Skanowanie i monitorowanie a bezpieczeństwo kontenerów w chmurze i środowisku programistycznym	344
Skanowanie obrazów kontenerów za pomocą programu Anchore Engine	345
Chef InSpec	349
Lokalny monitoring platformy Docker za pomocą natywnego narzędzia stats	350
Agregowanie danych monitoringowych w chmurze za pomocą narzędzia Datadog	353
Zabezpieczanie kontenerów w chmurze AWS	356
Alarmy bezpieczeństwa w usłudze AWS GuardDuty	357
Zabezpieczanie kontenerów w chmurze Azure	358
Monitorowanie kontenerów w chmurze Azure	358
Zabezpieczanie kontenerów przy użyciu usługi Security Center	359

Zabezpieczanie kontenerów w chmurze GCP	360
Analiza kontenerów i uwierzytelnienie binarne	361
Wykrywanie ataków za pomocą usługi Security Command Center	362
Podsumowanie	363
Dalsza lektura	364
Rozdział 16. Wnioski — koniec drogi, ale nie podróży	365
Wymagania techniczne	365
Kontenery w skrócie	366
Czego się dowiedziałeś o tworzeniu aplikacji	366
Wzorce projektowe	366
Poszerzenie wiedzy o tworzeniu i utrzymaniu aplikacji	369
Inżynieria chaosu i tworzenie niezawodnych systemów produkcyjnych	369
Bezpieczeństwo i dalsze kroki	371
Metasploit i testy penetracyjne	371
Podsumowanie	373

Wdrażanie i uruchamianie kontenerów w środowisku produkcyjnym

W miarę dojrzewania technologii kontenerowych i chmurowych zaczęło pojawiać się coraz więcej sposobów wdrażania kontenerów Docker. Niektóre z nich są proste i polegają na uruchamianiu kontenerów na pojedynczym komputerze, a inne bardziej zaawansowane, wykorzystujące takie funkcjonalności jak automatyczne skalowanie czy wielochmurowość. Kontenery można nawet uruchamiać bezpośrednio na fizycznym sprzęcie lub wykorzystując hybrydowe rozwiązania chmurowe.

W tym rozdziale poznasz różne opcje wdrożeniowe oraz związane z nimi kompromisy. Zbudujesz proste, ale w pełni funkcjonalne środowisko produkcyjne. Poznasz kryteria wyboru operatorów chmurowych, oferowane przez nich systemy zarządzania oraz korzyści płynące z uruchamiania kontenerów Docker we własnym środowisku oraz infrastrukturze hybrydowej. Co najważniejsze, będziesz potrafił podejmować świadome decyzje dotyczące wdrażania kontenerów Docker w środowisku produkcyjnym w zależności od różnych, często przeciwstawnych wymagań. Wiedza o spektrum dostępnych opcji pozwoli Ci podejmować lepsze decyzje.

W tym rozdziale opisane są następujące zagadnienia:

- uruchamianie kontenerów Docker w środowisku produkcyjnym,
- minimalne środowisko produkcyjne,
- zarządzane usługi chmurowe,
- tworzenie własnych klastrów Kubernetes,
- dobieranie właściwej konfiguracji produkcyjnej.

Wymagania techniczne

Do wykonania ćwiczeń opisanych w tym rozdziale będziesz potrzebował programu Git i platformy Docker. Jeżeli używasz systemu macOS lub Windows, zainstaluj program Docker Desktop (<https://www.docker.com/products/docker-desktop>) umożliwiający uruchamianie kontenerów na lokalnym komputerze. Zanim wybierzesz narzędzie do wdrażania kontenerów w środowisku produkcyjnym, musisz dowiedzieć się więcej o dostępnych opcjach.

W zależności od potrzeb może być konieczne założenie konta w usługach Amazon Web Services, Google Cloud, Microsoft Azure lub DigitalOcean. Większość z nich oferuje bezpłatne pakiety umożliwiające eksperymentowanie z różnymi opcjami przez określony czas. Wybierając najlepsze środowisko dla wdrażanej aplikacji, warto rozważyć kilka opcji. Jeżeli zarezerwujesz zasoby w chmurze, nie zapomnij ich usunąć, gdy przestaniesz z nich korzystać, aby nie spotkała Cię niemiła niespodzianka w postaci wysokiego rachunku. Większość środowisk chmurowych umożliwia określenie limitu, po przekroczeniu którego jest wysyłane powiadomienie. Jeżeli będziesz chciał się dowiedzieć więcej o bardziej skomplikowanych opcjach umożliwiających uruchamianie kontenerów Docker we własnej infrastrukturze lub bezpośrednio na sprzęcie (np. Openstack lub Packet, <https://www.packet.com>), będziesz potrzebował jednego lub kilku serwerów o odpowiednich parametrach.

Przykładowe kody wykorzystane w tym rozdziale są zapisane w katalogu *r05* w załączonych do książki plikach. Dodatkowo na stronie <https://bit.ly/2DYMria> znajduje się film instruktażowy (w języku angielskim).

Przykładowa aplikacja ShipIt Clicker

W dołączonych do książki plikach znajduje się kod prostej gry internetowej ShipIt Clicker, w której wiewiórka zachęca do wdrażania kontenerów w środowisku produkcyjnym. Im szybciej będziesz klikał, tym więcej wiewiórkodolarów zarobisz, za które w sklepie ShipIt Store będziesz mógł kupić prawo do wdrażania większej liczby kontenerów za każde kliknięcie lub do wdrażania ich bez klikania. Gra jest wyposażona w prosty interfejs HTML oraz interfejs REST API służący do komunikacji z bazą danych Redis, w której są przechowywane wyniki. Wersja gry wykorzystana w tym rozdziale jest prostym prototypem oferującym tylko część funkcjonalności pełnej wersji. Niemniej jednak ma wiele cech aplikacji produkcyjnej, goto-

wej do pierwszego wdrożenia. Można ją na przykład uruchamiać na kilku kontenerach za pomocą polecenia `docker-compose`. Aplikacja komunikuje się z przeglądarką gracza i serwerem Node.js za pomocą interfejsów Express i Swagger API, a w bazie Redis zapisuje wyniki i inne informacje.

Eksperymenty z grą ShipIt Clicker pozwolą Ci poznać bardziej zaawansowane aplikacje niż opisane w poprzednich rozdziałach. Zmieniając jej konfigurację i kod oraz stosując różnego rodzaju usługi i narzędzia, dowiesz się więcej o wdrażaniu aplikacji w środowisku produkcyjnym. W kolejnych rozdziałach poznasz inne sposoby wdrażania aplikacji, oferujące dodatkowe możliwości, ale kosztem ceny, złożoności i dostępności. Jednak zanim zaczniesz je stosować, dowiedz się więcej na ich temat.

Uruchamianie kontenerów Docker w środowisku produkcyjnym

Kontenery Docker można uruchamiać na lokalnym komputerze na wiele sposobów, ale proces ten jest bardzo prosty w porównaniu z mnogością opcji, jakie mają do dyspozycji programiści i administratorzy systemów w środowiskach produkcyjnych. Kilka największych firm informatycznych wykorzystuje platformę Docker lub inną technologię do masowego uruchamiania i konfigurowania kontenerów. Możliwość tworzenia samonaprawialnych klastrów, w których aplikacje działają poprawnie nawet w przypadku problemów z siecią lub sprzętem, przekonuje wiele organizacji do korzystania z platformy Docker. Entuzjazm jednak słabnie, gdy się okazuje, jak bardzo skomplikowane jest tworzenie klastrów odpornych na awarie.

Jednak nie trzeba wszystkiego robić samodzielnie. Wielu operatorów chmurowych oferuje usługi, dzięki którym uruchamianie aplikacji na platformie Docker jest prostsze. Jednym z rozwiązań, na które decyduje się wiele dużych organizacji, jest Kubernetes. Jest to projekt sponsorowany przez Google i rozwijany przez społeczność użytkowników, stanowiący alternatywę dla autorskich narzędzi kontenerowych. Rozwiązanie Kubernetes jest wynikiem doświadczenia, jakie firma Google nabyła podczas tworzenia i używania wewnętrznego narzędzia Borg do zarządzania kontenerami, które zdecydowała się publicznie udostępnić.

Jeżeli jednak zamierzasz uruchomić prostą aplikację internetową w najprostszej możliwej konfiguracji, nie musisz poznawać chmurowych narzędzi konfiguracyjnych, szczególnie gdy masz dostęp do serwera w internecie, na którym jest uruchomiona platforma Docker.

Minimalne środowisko produkcyjne

Platformę Docker można uruchamiać na szerokim spektrum sprzętu i systemów operacyjnych, znacznie różniących się między sobą zakresem dostępnego wsparcia oferowanego przez twórców platformy oraz zewnętrzne firmy. Platformę Docker można uruchamiać w różnych systemach operacyjnych, m.in. Linux, Apple macOS, Microsoft Windows, a nawet IBM S/390x.

Niezbędne minimum — Docker i Docker Compose na jednym hoście

Minimalne środowisko produkcyjne dla aplikacji opartej na platformie Docker składa się z jednego hosta, fizycznego lub wirtualnego, umożliwiającego korzystanie z narzędzia Docker Compose. Kilka najpopularniejszych systemów operacyjnych, m.in. Ubuntu LTS 16.04, 18.04, 20.04 oraz CentOS 7 i 8, posiada wbudowane określone wersje platformy. Dobrym wyborem mogą być wyspecjalizowane systemy przystosowane wyłącznie do uruchamiania kontenerów, na przykład CoreOS lub Container Linux. Są one jednak przeznaczone dla użytkowników znających inne, popularne systemy.

Produkcyjną aplikację opartą na platformie Docker można uruchomić nawet w systemie Windows lub macOS. Jednak w zależności od potrzeb i założonego ryzyka lepszym wyborem może okazać się system posiadający oficjalne wsparcie. Wszystko jest kwestią kompromisów.

Wsparcie dla platformy Docker

Bezpłatna wersja platformy Docker jest objęta wsparciem producenta przez bardzo ograniczony czas. Firma Docker Inc. (<https://www.docker.com>) co kwartał udostępnia wersję Community Edition (CE), dla której oferuje 4-miesięczny okres rozruchowy. Od 13 listopada 2019 r. wersja Enterprise Edition (EE) platformy Docker nosi nazwę Mirantis. Więcej informacji na jej temat można znaleźć na stronie <https://www.mirantis.com/company/press-center/company-news/mirantis-acquires-docker-enterprise>. Wersja EE jest objęta dłuższym okresem wsparcia, jest przystosowana do różnych wersji systemów Linux, Windows i macOS i wyposażona w więcej narzędzi do zarządzania. Szczegółowe informacje na ten temat są dostępne na stronie <https://docs.docker.com/ee>. Początkowo firma Mirantis zapowiedziała, że w listopadzie 2021 r. zakończy świadczenie wsparcia dla narzędzia Docker Swarm, wchodzącego w skład platformy Docker EE. Wycofała się jednak z tej decyzji w lutym 2020 r. (<https://devclass.com/2020/02/25/mirantis-to-keep-docker-swarm-buzzing-around-pledges-new-features>).

W pojedynku między narzędziami do zarządzania kontenerami zwycięzcą okazuje się Kubernetes, mimo że Mirantis wciąż utrzymuje produkt Docker Swarm.

Problemy z wdrażaniem na pojedynczym hoście

Uruchamianie kontenerów Docker na pojedynczym hoście ma wiele mankamentów. W przypadku poważnej awarii sprzętu lub oprogramowania albo problemów z połączeniem z internetem dostęp do aplikacji nie jest możliwy. Komputery są z założenia zawodne i nawet systemy korporacyjne wyposażone w rozwiązania zabezpieczające, takie jak zapasowe dyski, zasilacze i wentylatory, mogą ulegać awariom spowodowanym zewnętrznymi czynnikami. Dlatego warto stosować niezależne systemy monitoringu oraz systemy tworzenia i przywracania kopii zapasowych minimalizujących ryzyko. W takich sytuacjach należy rozważyć bardziej zaawansowane rozwiązania, na przykład systemy zarządzania oferowane przez innych producentów.

Zarządzane usługi chmurowe

Najprostszym sposobem uniknięcia problemów towarzyszących wdrażaniu aplikacji na pojedynczym hoście jest skorzystanie z zarządzanej usługi chmurowej oferującej rozwiązania do konfigurowania kontenerów. Najpopularniejsze tego rodzaju usługi to:

- Google Kubernetes Engine (GKE),
- Amazon Web Services Elastic Beanstalk (EB),
- Amazon Web Services Elastic Container Service (ECS),
- Amazon Web Services Elastic Kubernetes Service (EKS),
- Microsoft Azure Kubernetes Service (AKS),
- DigitalOcean Docker Swarm.

Większość z powyższych usług wykorzystuje do uruchamiania kontenerów narzędzie Kubernetes (<https://kubernetes.io>) opracowane przez Google. Przez wiele lat firma Google wykorzystywała do zarządzania kontenerami własny system Borg (<https://ai.google/research/pubs/pub43438>). Zainspirowana swoimi doświadczeniami postanowiła zbudować system dla użytku publicznego, któremu nadała nazwę Kubernetes.

W niektórych usługach chmurowych jest wykorzystywany system Docker Swarm, a w innych (m.in. AWS Elastic Beanstalk i AWS ECS) własne, niestandardowe systemy operatorów. Wszystkie umożliwiają programistom i administratorom zarządzanie serwerami z wieloma uruchomionymi kontenerami i opartymi na politykach mechanizmami rozmieszczania kontenerów w klastrze. Zadaniem tego rodzaju systemów jest uruchamianie i monitorowanie kontenerów oraz przenoszenie ich pomiędzy hostami w zależności od stanu i wymagań dotyczących skalowania. Wraz ze wzrostem popularności systemów do zarządzania kontenerami wielu operatorów, m.in. Google, Microsoft, Amazon Web Services i Digital Ocean, zaczęło oferować związane z nimi usługi, opisane w kolejnych podrozdziałach.

Google Kubernetes Engine

Google oferuje system Google Kubernetes Engine (GKE — <https://cloud.google.com/kubernetes-engine>) wykorzystujący narzędzie Kubernetes do zarządzania klastrem kontenerów w chmurze Google Cloud. Dzięki niemu nie trzeba samodzielnie utrzymywać i aktualizować głównych węzłów klastra. Węzły te nie są nawet widoczne w konsoli, ponieważ zarządza nimi bezpośrednio Google. Co więcej, operator nie obciąża klientów opłatami za utrzymywanie głównych węzłów. Jest to atrakcyjna oferta dla programistów, ponieważ pozwala uruchamiać klastry kontenerów niewielkim kosztem. Dodatkowy komfort zapewnia wsparcie techniczne świadczone bezpośrednio przez Google.

Google Cloud nie jest jednak największą ani nawet drugą co do wielkości usługą chmurową. Ponadto inne usługi oferowane przez Google nie są tak zróżnicowane jak Azure, AWS czy AliBaba.

Jeżeli korzystasz z usługi Google Cloud i potrzebujesz niedrogiego środowiska do eksperymentowania z narzędziem Kubernetes, zamierzasz używać go w środowisku produkcyjnym oraz nie jesteś związany z usługami innych operatorów chmurowych, przetestuj uruchamianie kontenerów w usłudze GKE.

AWS Elastic Beanstalk

Amazon oferuje usługę Elastic Beanstalk (<https://aws.amazon.com/elasticbeanstalk>) umożliwiającą uruchamianie aplikacji opartych na jednym lub kilku kontenerach Docker. W tym drugim przypadku jest wykorzystywana usługa ECS. Programiści korzystający z usługi Elastic Beanstalk mogą za pomocą narzędzia terminalowego łatwo wdrażać aplikacje w różnych środowiskach, a złożoność uruchamiania automatycznie skalowanych klastrów ukrywać w związanych plikach konfiguracyjnych.

Usługa Elastic Beanstalk jest prostsza w użyciu niż ECS lub EKS. Jest przeznaczona dla programistów potrzebujących szybko i przy nominalnym nakładzie pracy wdrażać aplikacje produkcyjne.

AWS ECS i Fargate

Amazon oferuje również system ECS (<https://aws.amazon.com/ecs>) do zarządzania kontenerami. System ten może pracować w dwóch trybach. W pierwszym kontenery są uruchamiane w instancjach EC2 zarządzanych przez właściciela konta, a w drugim w węzłach Fargate (<https://aws.amazon.com/fargate>) zarządzanych przez Amazon.

Korzystanie z systemu ECS opartego na instancjach EC2 lub węzłach Fargate jest uzasadnione wtedy, gdy wykorzystywane są inne usługi AWS. Dzięki niemu można wdrażać kontenery bez użycia narzędzi Kubernetes czy Docker Swarm. Ponieważ jest to własny system firmy Amazon, przenoszenie do niego kontenerów wymaga większej pracy niż w przypadku Kubernetes czy Docker Swarm. Ponadto trzeba się go nauczyć, ponieważ jest wyposażony w interfejs typowy dla usługi AWS, jak również wymaga umiejętności uruchamiania kontenerów Docker w tej usłudze.

AWS EKS

Amazon oferuje usługę EKS opartą na systemie Kubernetes, która jednak nie wymaga samodzielnego utrzymywania i konfigurowania głównych serwerów. Jest ona równoważna usłudze Google GKE. Jest ściśle zintegrowana z innymi usługami AWS. Pod względem uruchamiania głównych węzłów Kubernetes nie jest tak ekonomiczna jak GKE, ale ostateczne koszty uruchomienia w niej intensywnie wykorzystywanej aplikacji są niższe. Usługa EKS oparta na serwerze Kubernetes jest oferowana od 2018 r. Od tamtej pory firma Amazon usunęła wiele ograniczeń (na przykład brak możliwości definiowania strategii automatycznego skalowania aplikacji) i obecnie jest to niezwykle użyteczna usługa. W grudniu 2019 r. firma ogłosiła, że usługa EKS będzie

obejmowała również węzły Fargate. W ten sposób wzbogaciła usługę o elastyczność i łatwość zarządzania kontenerami.

Na początku 2020 r. oferta usług chmurowych Amazona była największa i najbardziej zróżnicowana. Jeżeli korzystasz już z usług AWS i chcesz pójść drogą sprawdzoną przez wielu innych użytkowników, rozważ EKS jako podstawową usługę do zarządzania kontenerami.

Microsoft Azure Kubernetes Service

Microsoft oferuje niezawodną usługę Azure Kubernetes Service (AKS) do zarządzania kontenerami. Jest ona szczególnie atrakcyjna dla użytkowników korzystających z wielu innych produktów Microsoftu, na przykład Windows, Visual Studio Code lub Active Directory. Ponadto rozwiązania programistyczne tej firmy są prostsze w użyciu niż oferowane przez innych producentów. Jeżeli jednak intensywnie korzystasz z wielu produktów Microsoftu, przedstawienie się na inne rozwiązania może być trudne.

Jeżeli chcesz szybko zacząć używać narzędzia Kubernetes ściśle zintegrowanego z Visual Studio Code, rozważ skorzystanie z usługi AKZ.

DigitalOcean Docker Swarm

Firma DigitalOcean oferuje usługę uruchamiania kontenerów w dość prostym systemie Docker Swarm. Powszechnie uważa się, że wdrażanie kontenerów za jego pomocą jest prostsze niż w przypadku Kubernetes czy nawet AWS ECS. Ponadto standardowo dostępne są w nim narzędzia platformy Docker.

Firma Mirantis po przejęciu platformy Docker przerwała świadczenie wsparcia dla systemu Docker Swarm i wznowiła je dopiero na żądanie użytkowników. Zważywszy na nierozstrzygnięte kwestie dotyczące wsparcia świadczonego przez głównego producenta, należy starannie rozważyć uruchamianie nowych aplikacji w systemie Docker Swarm.

Teraz gdy poznałeś różne możliwości uruchamiania kontenerów w środowiskach produkcyjnych, przyjrzyjmy się bliżej uruchamianiu aplikacji za pomocą narzędzi platformy Docker i systemu Kubernetes.

Tworzenie własnych klastrów Kubernetes

Jeżeli trzeba uruchomić aplikację we własnej infrastrukturze lub centrum danych albo w różnych usługach chmurowych, może się okazać konieczne zbudowanie własnego klastra. Gdy poznasz zalety i wady korzystania z platformy Docker i systemu Kubernetes we własnym środowisku lub hybrydowej infrastrukturze, będziesz mógł wybrać najlepsze rozwiązanie. Te opcje są bardziej skomplikowane niż korzystanie z wybranej zarządzanej usługi, ale mają kilka następujących zalet:

- Aktualizacja oprogramowania klastra może się odbywać według własnego harmonogramu z pełną kontrolą wersji używanej dzisiaj i jutro. Operatorzy chmurowi mogą instalować nowe wersje z opóźnieniem lub rezygnować z wersji, które zagrażają działaniu klastra.
- Można korzystać z jednego z wielu dojrzałych narzędzi, na przykład Kops, ułatwiających tworzenie klastrów k8s opartych na usłudze AWS EC2.
- Możliwe jest korzystanie z hybrydowych konfiguracji obejmujących centrum danych i środowisko chmurowe. Niektóre rozwiązania chmurowe, na przykład Cloud Anthos czy Azure Arc, oferują możliwość zarządzania środowiskami hybrydowymi, ale wiele innych nie ma tej funkcjonalności.
- Klastry Kubernetes można uruchamiać bezpośrednio na sprzęcie, bez dodatkowego obciążenia wprowadzanego przez hiperwizora.
- Można korzystać z platform nieobsługiwanych przez największych operatorów chmurowych, na przykład z klastra minikomputerów Raspberry Pi.
- Uzyskuje się pełną kontrolę nad infrastrukturą klastra, zintegrowaną z bazową platformą, na przykład OpenShift.
- Można uruchamiać własne prywatne rozwiązania chmurowe, takie jak OpenStack lub VMware Tanzu (wcześniej VMware Enterprise PKS).
- Można uruchamiać kontenery Docker w ramach bardziej zaawansowanej infrastruktury, na przykład Red Hat OpenShift lub Rancher, oferującej dodatkowe funkcjonalności, których nie ma system Kubernetes.

W praktyce korzystanie z powyższych rozwiązań jest bardziej skomplikowane niż z pojedynczego hosta lub zarządzanej klastrowej usługi Kubernetes oferowanej przez operatora chmurowego.

Dobieranie właściwej konfiguracji produkcyjnej

Z powodu ogromnej liczby dostępnych opcji wybranie właściwej metody wdrożenia aplikacji może być trudnym zadaniem wymagającym uwzględnienia następujących czynników:

- **Konfiguracja:** jak trudne jest przejście z lokalnego środowiska programistycznego do produkcyjnego?
- **Funkcjonalności:** jakie się dostępne możliwości wdrażania, testowania, monitorowania, alarmowania i raportowania?
- **Koszty:** jak wysokie są opłaty początkowe i miesięczne?
- **Wsparcie:** czy jest dostępne wsparcie operatora lub społeczności użytkowników?
- **Elastyczność:** czy jest możliwość automatycznego lub ręcznego skalowania aplikacji wraz ze wzrostem obciążenia?
- **Dostępność:** czy aplikacja będzie dostępna w przypadku awarii usług, hostów i sieci?
- **Unikatowość:** czy trudno będzie zmienić strategię wdrażania aplikacji?

Uruchamianie kontenerów Docker na pojedynczym hoście jest niedrogim i prostym rozwiązaniem, jednak niezadowalającym pod względem skalowania i niezawodności. Wszystkie usługi chmurowe wykorzystujące system Kubernetes są dobrze zaprojektowane pod względem skalowania i niezawodności, ale trudniejsze w konfiguracji i użytkowaniu. Z kolei systemy inne niż Kubernetes są na swój sposób unikatowe. Uruchamianie własnych klastrów w chmurze, bezpośrednio na sprzęcie lub w środowisku hybrydowym zapewnia ogromną elastyczność, ale kosztem większej złożoności i trudności utrzymania.

Świadomość zalet i wad poszczególnych systemów pomaga w wyborze odpowiednich technologii wdrażania aplikacji. Poniższa tabela przedstawia moje subiektywne oceny w skali od 1 (najgorsza) do 5 (najlepsza) i porównuje poszczególne technologie.

Technologia	Konfiguracja	Funkcjonalność	Koszty	Wsparcie	Elastyczność	Dostępność	Unikatowość
Pojedynczy host	5	1	5	3	1	1	5
Google GKE	4	5	3	4	5	5	3
AWS EB	5	3	3	4	5	5	3
AWS ECS	2	2	3	4	5	5	2
AWS EKS	4	5	2	5	5	5	3
Microsoft AKS	4	4	3	4	5	5	2
DigitalOcean Docker Swarm	4	2	3	1	4	4	2
Własny klaster k8s (sprzęt)	2	4	3	2	1	3	3
Własny klaster k8s (chmura publiczna)	3	5	2	2	5	5	2
Własny klaster k8s (infrastruktura hybrydowa)	1	5	3	2	5	5	4
OpenShift	3	4	1	5	4	4	1
OpenStack i Kubernetes	1	5	1	1	4	4	2

Powyższą tabelę możesz wykorzystywać do porównywania poszczególnych rozwiązań. Oceniając dwie lub więcej opcji, będziesz miał lepsze wyobrażenie o tym, która z nich będzie bardziej odpowiednia. Aby wykorzystać oceny zawarte w powyższej tabeli, przygotuj osobną tabelę i porównaj alternatywne rozwiązania. Przypisz poszczególnym cechom priorytety, od najwyższego (5) do najniższego (1), a następnie pomnóż je przez oceny z tabeli. W ten sposób uzyskasz ostateczną ocenę.

W poniższej przykładowej tabeli wysokie priorytety ma łatwość konfiguracji, minimalizacja kosztów i minimalizacja unikatowości, natomiast niskie ma dostępność i elastyczność. Są to typowe priorytety dla wielu nowych i praktycznych aplikacji. Często twórcy aplikacji chcą szybko uzyskać zamierzony efekt, nawet kosztem rezygnacji z innych cech. Liczby zawarte w kolumnach *Opcja 1* i *Opcja 2* są ilorazami priorytetów i ocen z powyższej tabeli.

Cecha	Priorytet	Opcja 1: pojedynczy host	Opcja 2: Google GKE
Konfiguracja	5	25	20
Funkcjonalności	1	1	5
Koszty	5	25	15
Wsparcie	3	9	12
Elastyczność	1	1	5
Dostępność	1	1	5
Unikatowość	4	16	12
Razem		78	74

W tym przypadku opcja 1, pojedynczy host, uzyskała wyższą ocenę. Ważna była łatwość konfiguracji, niskie koszty i unikatowość, dlatego te cechy uzyskały wyższe oceny w porównaniu z innymi. Na podstawie uzyskanej ostatecznej oceny można podjąć decyzję o wyborze opcji. Zwróć uwagę, że gdyby priorytety dostępności lub elastyczności były nieznacznie wyższe, lepszą ostateczną ocenę uzyskałaby opcja 2, Google GKE.

Może się okazać, że konfiguracja hybrydowa również spełniłaby Twoje oczekiwania i problem można by rozwiązać za pomocą różnych opcji. Na przykład w przypadku normalnego, codziennego obciążenia lepszy byłby klaster we własnej infrastrukturze, który podczas chwilowego wzrostu można by było rozbudowywać o rozwiązania chmurowe.

Ćwiczenie — dołącz do zespołu ShipIt Clicker

Wyobraź sobie, że dołączyłeś właśnie do zespołu rozwijającego aplikację ShipIt Clicker. Jego członkowie opracowali już podstawowy szkielet gry (opisany w załączonym do książki dokumencie *ShipIt_Clicker-spec.md*) i zbudowali prototyp oferujący minimalne funkcjonalności niezbędne do budowania, testowania i pakowania aplikacji w kontenerach Docker.

Członkowie zespołu są ekspertami od projektowania i programowania interfejsu i serwerów, ale nie wiedzą, jak się wdraża aplikacje w środowisku produkcyjnym. Twoje doświadczenie w tej dziedzinie jest znacznie większe. Pliki *Dockerfile* i *docker-compose.yml* są przygotowane i sprawdzone.

Aby dowiedzieć się, jak działa aplikacja ShipIt Clicker w wersji przygotowanej w tym rozdziale, uruchom ją na lokalnej stacji roboczej.

Do uruchomienia kontenerów użyj polecenia `docker-compose up`. W ten sposób będziesz mógł ocenić opcje wdrożeniowe i poeksperymentować ze zmianami w celu przygotowania aplikacji do wdrożenia w środowisku produkcyjnym. Uzyskasz wynik podobny do pokazanego niżej. W dalszej części rozdziału znajdziesz szczegółowe objaśnienia do każdej grupy wierszy.

```
$ docker-compose up
Building shipit-clicker-web
Step 1/11 : FROM ubuntu:bionic
----> 775349758637
Step 2/11 : RUN apt-get -qq update && apt-get -qq install
-y nodejs npm > /dev/null
----> Using cache
----> f8a9a6eddb8e
```

Powyższy wynik zawiera informację, że został wykorzystany obraz `ubuntu:bionic`, a następnie zainstalowane pakiety systemu operacyjnego.

W krokach 3 – 5 pliku *Dockerfile* jest przygotowywany obraz kontenera do instalacji aplikacji. W tym celu są tworzone najważniejsze katalogi i kopiowane pliki konfiguracyjne:

```
Step 3/11 : RUN mkdir -p /app/public /app/server
----> Using cache
----> f7e56a628e8b
Step 4/11 : COPY src/package.json* /app
----> eede94466dc7
Step 5/11 : WORKDIR /app
----> Running in adcadb6616c2
Removing intermediate container adcadb6616c2
----> 6256f613803e
```

Następnie są instalowane moduły:

```
Step 6/11 : RUN npm install > /dev/null
----> Running in 02ae124cf711
npm WARN deprecated superagent@3.8.3: Please note that v5.0.1+
of superagent removes User-Agent header by default, therefore
you may need to add it yourself (e.g. GitHub blocks requests
without a User-Agent header). This notice will go away with
v5.0.2+ once it is released.
npm WARN optional Skipping failed optional dependency /
chokidar/fsevents:
npm WARN notsup Not compatible with your operating system or
architecture: fsevents@1.2.11
npm WARN shipit-clicker@1.0.5 No repository field.
npm WARN shipit-clicker@1.0.5 No license field.
Removing intermediate container 02ae124cf711
----> 64ea4b348ed1
```

W kolejnych krokach w obrazie kontenera są umieszczane następne pliki konfiguracyjne i pliki źródłowe:

```
Step 7/11 : COPY src/.babelrc src/.env src/.
nodemonrc.json /app/
----> 88e88c1bc35d
```

```
Step 8/11 : COPY src/public/ /app/public/
----> c9872fcccc1c9
Step 9/11 : COPY src/server/ /app/server/
----> f6e76811659a
```

Na koniec udostępniany jest port i uruchamiana aplikacja:

```
Step 10/11 : EXPOSE 3000
----> Running in 75fbd217ef27
Removing intermediate container 75fbd217ef27
----> 03faaa0e8030
Step 11/11 : ENTRYPOINT DEBUG='shipit-clicker:*' npm run dev
----> Running in 0a44ab13b0d3
Removing intermediate container 0a44ab13b0d3
----> ab6e4da773e7
Successfully built ab6e4da773e7
```

Tak zbudowanemu kontenerowi jest przypisywany znacznik latest:

```
Successfully tagged chapter5_shipit-clicker-web:latest
WARNING: Image for service shipit-clicker-web was built because
it did not already exist. To rebuild this image you must use
`docker-compose build` or `docker-compose up --build`.
```

Siła polecenia `docker-compose up` ujawnia się w poniższym wyniku. Jedno polecenie, użyte na samym początku, nie tylko tworzy kontener aplikacji, ale również uruchamia wszystkie kontenery; tutaj najpierw aplikacyjny, a następnie bazę Redis. Kontener Redis podczas uruchamiania wysyła szczegółowe informacje, które polecenie `docker-compose up` umieszcza w swoich komunikatach:

```
Starting chapter5_redis_1 ... done
Creating chapter5_shipit-clicker-web_1 ... done
Attaching to chapter5_redis_1, chapter5_shipit-clicker-web_1
redis_1          | 1:C 04 Feb 2020 06:15:08.774 #
o000o000o000o Redis is starting o000o000o000o
redis_1          | 1:C 04 Feb 2020 06:15:08.774 # Redis
version=5.0.7, bits=64, commit=00000000, modified=0, pid=1,
just started
redis_1          | 1:C 04 Feb 2020 06:15:08.774 # Warning:
no config file specified, using the default config. In order to
specify a config file use redis-server /path/to/redis.conf
redis_1          | 1:M 04 Feb 2020 06:15:08.776 * Running
mode=standalone, port=6379.
redis_1          | 1:M 04 Feb 2020 06:15:08.776 # 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 04 Feb 2020 06:15:08.776 # Server
initialized
```

Zwróć uwagę na komunikaty informujące, że jądro systemu Linux nie jest dokładnie przystosowane do uruchomienia bazy Redis. Jest to przykład sytuacji, w której platforma Docker nie zapewnia uzyskania optymalnych wyników (choć są one wystarczające):

```

redis_1 | 1:M 04 Feb 2020 06:15:08.776 # 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 04 Feb 2020 06:15:08.776 * DB
loaded from disk: 0.000 seconds
redis_1 | 1:M 04 Feb 2020 06:15:08.776 * Ready to
accept connections

```

W tym momencie baza Redis jest gotowa. Następnie polecenie `docker-compose` uruchamia kontener ShipIt Clicker, wykorzystując polecenie widoczne w pokazanym wcześniej wyniku `ENTRYPOINT DEBUG('shipit-clicker:*' npm run dev)`:

```

shipit-clicker-web_1 |
shipit-clicker-web_1 | > shipit-clicker@1.0.5 dev /app
shipit-clicker-web_1 | > nodemon server --exec babel-node
--config .nodemonrc.json | pino-pretty
shipit-clicker-web_1 |
shipit-clicker-web_1 | [nodemon] 1.19.4
shipit-clicker-web_1 | [nodemon] to restart at any time, enter
`rs`
shipit-clicker-web_1 | [nodemon] watching dir(s): *.*
shipit-clicker-web_1 | [nodemon] watching extensions:
js,json,mjs,yaml,yml
shipit-clicker-web_1 | [nodemon] starting `babel-node server`
shipit-clicker-web_1 | [1580796912837] INFO (shipit-
clicker/47 on 52e6d59c6121): Redis connection established
shipit-clicker-web_1 |   redis url: "redis://redis:6379"
shipit-clicker-web_1 | [1580796913083] INFO (shipit-
clicker/47 on 52e6d59c6121): up and running in development @:
52e6d59c6121 on port: 3000}

```

Po wykonaniu powyższych operacji możesz zacząć grę, otwierając w przeglądarce stronę <http://localhost:3000>. Rysunek 5.1 przedstawia główne menu oraz odnośnik do dokumentacji interfejsu API, dostępnej pod adresem <http://localhost:3000/api-explorer>.

Po uruchomieniu aplikacji i sprawdzeniu, jak działa, możesz poznać różne sposoby jej wdrażania.

Ćwiczenie — wybór właściwej metody wdrożenia

Konfiguracja opisana w tym rozdziale umożliwia uruchamianie aplikacji w lokalnym środowisku programistycznym. Ma ona jednak pewne mankamenty, które mogą przysparzać problemów przy wdrażaniu aplikacji w środowisku produkcyjnym.



Rysunek 5.1. Główne menu gry Shiplt Clicker

Uczestnikami gry na etapie tworzenia prototypu są:

- inni twórcy gry i zarząd firmy,
- szerokie grono entuzjastów biorących udział w programie Alfa,
- profesjonalni testerzy po drugiej stronie kuli ziemskiej.

Zarząd chce jak najszybciej udostępnić grę testerom ochotnikom i zawodowcom. Chce również znać dostępne opcje i koszty wdrożenia w środowisku produkcyjnym, w którym będzie można skalować grę, w miarę jak będzie zdobywała popularność, a inwestorzy zgodzą się na kampanię reklamową mającą na celu pozyskanie subskrybentów.

Ponieważ znasz platformę Docker i różne opcje wdrażania aplikacji, dostałeś następujące zadania:

- przygotowanie tabeli ocen różnych opcji i przedstawienie zarządowi wstępnego środowiska wdrożeniowego,
- przedstawienie zarządowi innych opcji, zapewniających większą elastyczność i skalowalność aplikacji,
- przygotowanie na podstawie cen usług różnych operatorów arkusza zawierającego jednorazowe i miesięczne koszty na najbliższy rok.

Rozwiązanie

Porównaj tabelę, którą uzyskałeś, z opisaną w podrozdziale „Dobieranie właściwej konfiguracji produkcyjnej” i przeanalizuj różnice. Pokaż model kosztów i tabelę decyzyjną kolegom. Zapytaj ich, co by wybrali i czy zgadzają się z Twoją decyzją.

Ćwiczenie — ocena plików Dockerfile i docker-compose.yml

Zarząd potrzebuje Twojej pomocy w usprawnieniu wdrożenia produkcyjnego i określeniu obszarów, w których można byłoby go poprawić. Interesują go następujące kwestie:

- Czy ustawienia określone w plikach *Dockerfile* i *docker-compose.yml* są odpowiednie dla aplikacji?
- Jakie ustawienia można zmienić, aby lepiej przygotować aplikację do wdrożenia w środowisku produkcyjnym?
- Jaki wpływ na kontenery ma dystrybucja systemu operacyjnego wskazana w instrukcji FROM?

Rozwiązanie

Przejrzyj zawartość plików *Dockerfile* i *docker-compose.yml* zapisanych w katalogu *r06* i sprawdź, czy Twoje zalecenia są z nimi zgodne. Tymi plikami zajmiemy się dokładniej w rozdziale 6., „Wdrażanie aplikacji przy użyciu Docker Compose”.

Po wykonaniu ćwiczeń podsumujmy, czego dowiedziałeś się na temat możliwości wdrażania dekoratorów Docker w środowisku produkcyjnym.

Podsumowanie

W tym rozdziale poznałeś różne możliwości wdrażania kontenerów Docker w środowisku produkcyjnym. Przekonałeś się, że wiele z nich wymaga kompromisów. Zbudowałeś najmniejsze środowisko produkcyjne. Poznałeś oferty różnych operatorów chmurowych, ich systemy do zarządzania kontenerami oraz korzyści płynące z uruchamiania kontenerów we własnej infrastrukturze i środowisku hybrydowym. Wiesz już, jak na podstawie stawianych wymagań podejmować decyzje dotyczące wdrażania kontenerów Docker.

Nabytą wiedzę możesz wykorzystać do wdrażania aplikacji w prawdziwym środowisku produkcyjnym. Dokładna znajomość różnych technologii jest bardzo ważna, ponieważ każda strategia ma swoje zalety i wady. W przyszłości może być potrzebne automatycznie skalujące się środowisko, ale dzisiaj wystarczy takie, w którym aplikacja będzie po prostu działać.

W następnym rozdziale dowiesz się, jak wdrażać kontenery Docker na pojedynczym hoście, zachowując możliwość rozwijania ich na lokalnej stacji roboczej.

PROGRAM PARTNERSKI

— GRUPY HELION —

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

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA
Helion 

Docker. Wszechstronne wdrożenia w najlepszym stylu

Docker zyskuje coraz większe uznanie programistów. Dzięki swojej niezależności od platformy kontenery pozwalają na uruchamianie kodu w różnych środowiskach, zarówno w centrach danych, jak i w chmurze. Zastosowanie kontenerów bardzo upraszcza opracowanie, testowanie, wdrażanie i skalowanie aplikacji. Pozwala też na automatyzację przepływu pracy i stałe doskonalenie aplikacji. Aby skorzystać ze wszystkich tych zalet, architekci, projektanci i programiści muszą wszechstronnie i głęboko poznać wiele różnych aspektów zarządzania środowiskiem kontenerowym.

Oto wyczerpujące omówienie wszystkich zagadnień niezbędnych do tworzenia i rozwijania aplikacji w Dockerze. W książce zaprezentowano różne metody wdrażania i uruchamiania kontenerów, pokazano również, jak wykorzystuje się je w środowisku produkcyjnym. Wskazano właściwe techniki używania narzędzi Jenkins, Kubernetes i Spinnaker. Przedstawiono metody monitorowania, zabezpieczania i skalowania kontenerów za pomocą takich narzędzi jak Prometheus i Grafana. Nie zabrakło opisu wdrażania kontenerów w różnych środowiskach, między innymi w chmurowej usłudze Amazon Elastic Kubernetes Service, a także — na koniec — kwestii bezpieczeństwa Dockera i związanych z tym dobrych praktyk.

W książce:

- gruntowne wprowadzenie do Dockera i programowania w VirtualBox
- tworzenie systemów z kontenerów i ich wdrażanie w środowisku produkcyjnym
- ciągłe wdrażanie oprogramowania
- skalowanie, testy obciążeniowe i zagadnienia bezpieczeństwa
- stosowanie zewnętrznych narzędzi: AWS, Azure, GCP i innych

Richard Bullington-McGuire od ponad ćwierćwiecza jest architektem oprogramowania i praktykiem DevOps. Aktywnie korzysta z Dockera. Jest członkiem IEEE i ACM, posiada sześć certyfikatów AWS.

Andrew K. Dennis od blisko dwóch dekad jest architektem systemów bezpieczeństwa. Organizuje konferencję Security BSides CT.

Michael Schwartz od ponad czterdziestu lat jest architektem systemów. Rozwijał pierwsze gry wideo, a jego najnowszy projekt to RoboDomo, system domowej automatyki.

Helion 	<i>Sprawdź nasze szkolenia!</i>	KOD KORZYŚCI Sięgnij po więcej! ▶	
 helion.pl	 AKADEMIA IT & BUSINESS	ISBN 978-83-283-7727-1	
 0 801 339900			
 0 601 339900	WWW.SZKOLENIA.HELION.PL	9 788328 377271	
INFORMATYKA W NAJLEPSZYM WYDANIU		Cena: 79,00 zł	

Packt